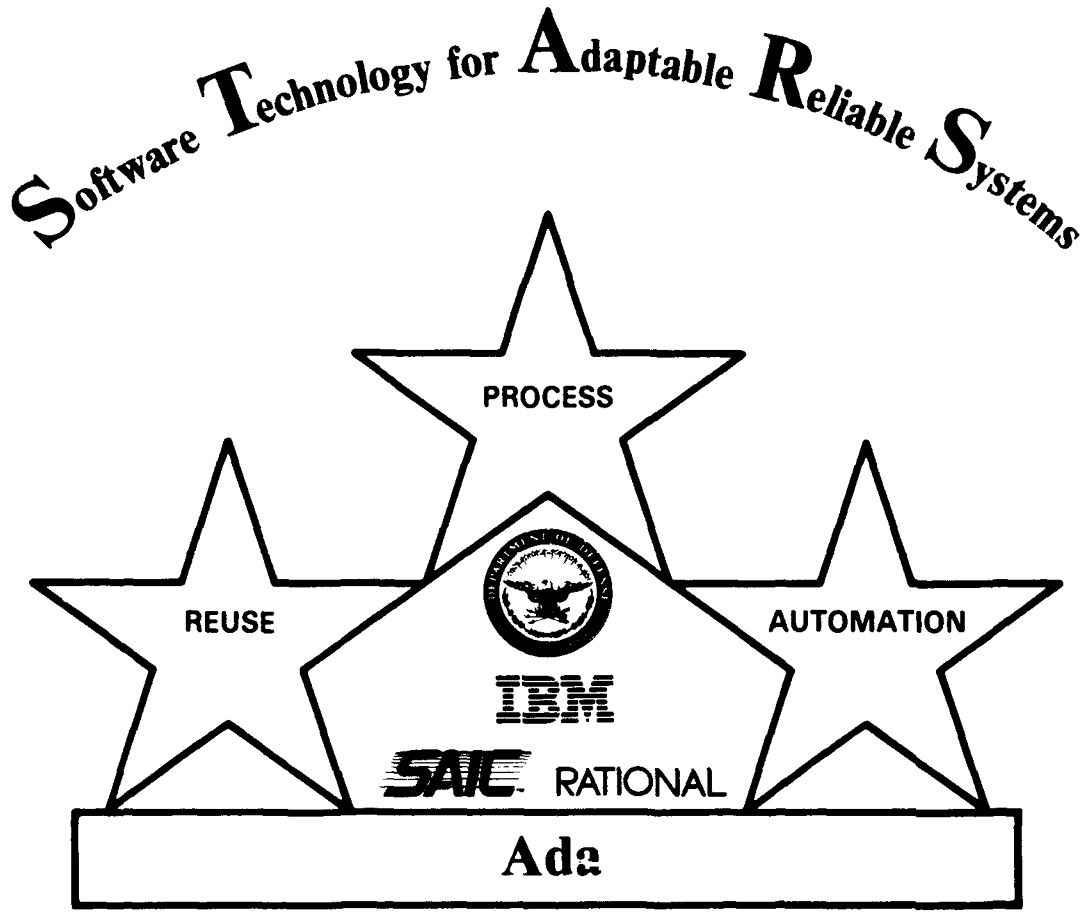
Document No. 0110-001
17 March 1989

2

Environment Capability Matrix

for the



Contract No. F19628-88-D-0032

CDRL Sequence No. 0110

17 March 1989

REPORT DOCUMENTATION PAGE		Form Approved OMB No. 0704-0188

[illegible]

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE March 17, 1989	3. REPORT TYPE AND DATES COVERED Final
4. TITLE AND SUBTITLE Environment Capability Matrix		5. FUNDING NUMBERS C: F19628-88-D-0032
6. AUTHOR(S)		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) IBM Federal Sector Division 800 N. Frederick Avenue Gaithersburg, MD 20879		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Electronic Systems Division Air Force Systems Command, USAF Hanscom AFB, MA 01731-5000		10. SPONSORING/MONITORING AGENCY REPORT NUMBER CDRL Sequence No. 0110
11. SUPPLEMENTARY NOTES		
12a. DISTRIBUTION/AVAILABILITY STATEMENT		12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This report presents the results of an analysis of four key Ada software engineering environments -- the Ada Language System / Navy (ALS/N), the Rational R1000 Development System, the Software Development and Maintenance Environment (SDME) and the Space Station Software Support Environment (SSE). The study recommends that the STARS SEE incorporate such capabilities as software process management, standard interfaces, data base support, and software engineering support, including support for Ada, multi-target development, prototyping, and reuse.

14. SUBJECT TERMS STARS, software engineering environments, ALS/N, SDME		15. NUMBER OF PAGES 63
		16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)

Environment Capability Matrix
for the
Software Technology for Adaptable, Reliable Systems (STARS) Program

Contract No. F19628-88-D-0032

CDRL Sequence No. 0110

17 March 1989

Prepared for:

Electronic Systems Division
Air Force Systems Command, USAF
Hanscom AFB, MA 01731-5000

Prepared by:

IBM Systems Integration Division
18100 Frederick Pike
Gaithersburg, MD 20879

Abstract

This document, IBM Contract Data Requirements List (CDRL) Sequence Number 0110, Environment Capability Matrix, presents the IBM Environment Capability Matrix and an assessment of key Ada environments as called for in the IBM Q3 Statement of Work (SOW).

In analyzing the environments, we found that most supported traditional environment capabilities such as configuration management and code development, but do not support more recently identified capabilities such as reuse and prototyping.

Based on the analyses, we recommend the SEE incorporate several capabilities including:

- Central Database,
- Standard Interfaces,
- Software Process Manager, and
- Software Engineering Support including
  - Ada Support,
  - Multi-Target Support,
  - Prototyping Support, and
  - Reuse Support.

This document results from the work of the IBM Q3 team of Ed Matthews and Mary Catherine Ward.

Table of Contents

Introduction

Scope

This document, IBM Contract Data Requirements List (CDRL) Sequence Number 0110, Environment Capability Matrix, presents the IBM Environment Capability Matrix and an assessment of key Ada environments as called for in the IBM Q3 Statement of Work (SOW).

Task Description

This CDRL analyzes and assesses four key Ada environments:

- the Ada Language System/Navy (ALS/N),
- the Rational[1] R1000 Development System (Rational),
- the Software Development and Maintenance Environment (SDME), and
- the Space Station Software Support Environment (SSE).

Because of time and resource constraints we could not install these environments for analysis. Instead, the documents detailing the requirements and specifications for each environment became our basis for analysis. We reviewed the documentation and mapped the environment capabilities against a derived software environment taxonomy. The results are presented in the IBM Environment Capability Matrix.

We also summarized and assessed each environment's contributions in the following areas:

- operational concept,
- foundation capabilities,
- user interface,
- software engineering capabilities, and
- reusability support.

Based on our analyses we make recommendations for capabilities the STARS Software Engineering Environment (SEE) should provide.

Relationship to Other Q3 CDRLs

CDRL 0100[2] , the predecessor to this CDRL, contains a taxonomy of environment capabilities derived from existing taxonomies and augmented with the capabilities that define the Software First Life Cycle (SFLC). This taxonomy fills the vertical axis of the Environment Capability Matrix.

1 Rational is a registered trademark and Rational Environment is a trademark of Rational. .

2 See (IBM0100). Following the model established in this footnote, we place references to other documents in parentheses. We include full citations in "Acronyms" on page 18.

The contents of this CDRL serve as input to CDRL 0090[3] which makes recommendations for future STARS environment activities.

[3] See (IBM0090).

Environment Capabilities

During our study of software engineering environments over the past few years, we noticed that the functions and capabilities of a given environment typically number in the hundreds, and given a sufficient level of detail, could potentially number in the thousands. It would be difficult to understand and comprehend this number of capabilities without some means of data reduction. A classification scheme that can be examined in varying levels of detail, such as a taxonomy, provides a means of abstraction of environment capabilities. We concluded that for maximum understanding of the STARS Software Engineering Environment, we needed to provide such a classification scheme. We called it the STARS SEE Environment Capability Taxonomy and set about defining it.

STARS SEE Environment Capability Taxonomy

We realized that we could not define a taxonomy without first having defined the Software First Life Cycle (SFLC), so we drafted the definition documented in section "Appendix B. Capability Definitions" on page 29. Having no experience with the SFLC, we needed empirical evidence that this life cycle can work. Although the taxonomy definition was not a software development effort, we set about doing it in a way which mimics the SFLC. Two of the key technologies espoused by the SFLC are prototyping and reuse. Consistent with gaining empirical evidence about the feasibility of these two technologies, we decided to produce a prototype of the Environment Capability Taxonomy by reusing previous taxonomy work. Once this prototype taxonomy had been developed, we evaluated it, and, unsatisfied, iterated through taxonomy development to arrive at a new prototype. We continued in this fashion until we arrived at a taxonomy that would serve our purposes, learning valuable lessons about reuse and prototyping that we could apply to software development and the SFLC.

We started composing our prototype taxonomy from one described by (Kean85) and (Hough82). This taxonomy (hereinafter "NBS Taxonomy") is organized by classes of tool features (i.e., management, transformation, and analysis classes) and within these classes by life cycle phase. The following partial taxonomy is similar in organization to the NBS Taxonomy:

```
Management
  Coding
    (management capabilities for the coding phase)
  Testing
    (management capabilities for the testing phase)
Analysis
  Coding
    (analysis capabilities for the coding phase)
  Testing
    (analysis capabilities for the testing phase)
```

In evaluating the NBS Taxonomy, we had difficulty adopting it as our baseline. Our initial reaction was that it did not describe a software engineering environment in terms that made it easy for us to comprehend the life cycle supported by that environment. We decided that to comprehend the environment and life cycle, we needed to invert the tool feature classes and the life cycle phases like the following:

```
Coding
  Management
     (management capabilities for the coding phase)
  Analysis
     (analysis capabilities for the coding phase)
```

We asked six technical persons and two managers to refute our initial reaction to the NBS taxonomy by indicating which of the two candidate organizations they preferred. All eight persons indicated that the second, inverted organization was more understandable.

Further examination of the NBS Taxonomy revealed that it is based on the traditional DoD-STD-2167 waterfall life cycle model[4] and that it is not concerned with certain ancillary environment capabilities (e.g., user account management) that operating systems have traditionally performed. Also, the latest version of the NBS taxonomy is specifically multi-lingual rather than Ada-oriented, and does not use object-oriented terminology.

For the initial prototype STARS Environment Capability Taxonomy, we modified the NBS Taxonomy to invert tool feature classes and life cycle phases and to embrace Ada, object-orientation, the Software First Life Cycle, and certain ancillary capabilities.

The resultant hybrid taxonomy had large gaps, so we subjected it to another iteration and augmented it with portions of other taxonomies (E&VCS87), (E&VRM88), most notably that used by the Software Engineering Institute in evaluating Ada environments (Weide87). We also borrowed capabilities from other sources (Dart87), (Lyons86), and (Pened88). In evaluating the resultant prototype taxonomy, we compared its capabilities to those of existing environments. As we discovered unique and desirable capabilities during analysis of the environments, we added them to the taxonomy, creating new prototype versions closer to meeting our needs.

One concern with our interim taxonomies was that, like the NBS Taxonomy, they were extraordinarily lengthy. We found that one way to shrink the taxonomy without removing any environment capability detail was to remove the management, analysis, and transformation sublevels. So, a partial taxonomy that resembled:

```
Testing
  Management
     (management capabilities for the testing phase)
  Analysis
     (analysis capabilities for the testing phase)
```

shrank to:

```
Testing
  (capabilities for testing)
```

Also to reduce the taxonomy length (at the risk of being less complete) we eliminated repetition of identical capability and marked this with references to those sections of the taxonomy where the capability could be found. To further reduce the taxonomy size, in the hope that succinctness promotes comprehension, those aspects which permeate all phases of the life cycle (primarily management capabilities) have been factored out into their own sections.

The taxonomy which fills the vertical axis of the Environment Capability Matrix is the result of many iterations and represents an application of the prototyping and reuse technologies that the SFLC espouses.

[4] The waterfall is shown in Figure 1 of DoD-STD-2167 (DoD2167).

Environment Capability Matrix

The Environment Capability Matrix which maps the environments against the STARS SEE Environment Capability Taxonomy appears in "Appendix A. Environment Capability Matrix" on page 21. This matrix is an integral part of this CDRL detailing the environments' capabilities. It is the basis for the summaries and assessments of the next section.

Note: To establish a common working vocabulary we define each capability appearing in the matrix. The definitions appear in "Appendix B. Capability Definitions" on page 29.

Environment Analysis

The following section summarizes and assesses the capabilities of the environments studied. As part of the analysis we classify each environment according to the classifications defined in (Sterl88). We list them here for reference:

Toolkit Environments consist of a collection of smaller tools which may or may not share information with one another.

Language-Centered Environments provide integrated tools supporting a specific language.

Environment Frameworks can create a variety of environments, each tailored to the needs of a particular software development project.

The assessments reference capabilities whose definitions can be found in the section "Appendix B. Capability Definitions" on page 29.

Assessments

Ada Language System/Navy (ALS/N)

Background

Softech developed the ALS/N for the U.S. Navy based on a system specification document written by the Naval Ocean Systems Center. This document, completed in 1986, defines the ALS/N as a Minimal Ada Programming Support Environment (MAPSE) and as the foundation of the Navy's Software Engineering Environment.

We used (ALS/N86) and (Weide87) in analyzing the ALS/N.

Key Capabilities

Operational Concept: Although some of the ALS/N tool set is integrated, the ALS/N is generally a non-integrated toolkit environment.

Foundation Capabilities: An Entity-Relationship-Attribute[5] (ERA) database is the core of the ALS/N foundation. Entities (objects in ALS/N terminology) represent objects in the environment; relations (associations in ALS/N terminology) represent an association between entities; and, attributes represent properties of entities. This data model supports configuration management, access control, library management, and may support additional environment capabilities. In addition, the ALS/N has a layered architecture for portability.

[5] See (Chen76).

User Interface: A modified Ada syntax command language is the interface for accessing tools of the ALS/N tool set. The user interface provides little command assistance (e.g., command completion, aliasing, prompting). The help facility is minimal in scope and not context sensitive.

Software Engineering Capabilities: ALS/N supports program development (i.e., edit, compile, link) well. It also provides text editing (screen and line), browsing, formatting, and comparison capabilities.

The multi-targeted Ada compiler has a machine independent front end that generates a DIANA intermediate language representation of the code. The machine dependent middle pass transforms the DIANA into a form suitable for the target-specific code generator. The ALS/N provides multiple target support including target linkers, debuggers, and profilers.

A well-integrated program library manager represents compilation units as trees in the ERA model database, giving both programmers and other tools access to program library information. The library manager provides the capability to query unit information and statistics, recompilation status, completeness status, and interdependency information. Through the library manager, a programmer may acquire a link to previously compiled units. When portions of a program are previously compiled, the compiler will generate code only for those portions that are new.

The command procedures of ALS/N provide a simple form of process management. Users invoke the command procedures defined by the system administrator via a command name. As specified in (ALS/N86), "a command procedure handles all complexities of tool invocation, and sequencing, database management, error recovery, and collection of development statistics with no need for the users' involvement or awareness."

Reusability Support: The ALS/N does not explicitly support reuse.

Other Capabilities: The most extensive management capabilities that ALS/N supports are system management and configuration management. Both are tightly coupled with the ALS/N database. For example, database attributes provide information necessary for discretionary and mandatory access control. Likewise the configuration management functions rely heavily on the database to provide information for configuration, version, and product release control.

Rational R1000 Development Environment (Rational)

Background

The R1000 Development Environment is a commercially available Ada development facility developed by Rational.

We used (Ratio85), (Ratio87), and (Ratio88) in analyzing the Rational.

Key Capabilities

Operational Concept: The Rational is a well-integrated, language-centered environment supporting the Ada programming language.

Foundation Capabilities: Rational's proprietary database permits extensive integration capabilities which make the environment appear seamless to users.

User Interface: The capability to invoke Item-Operation sequences makes the Rational user interface object-oriented. In Item-Operation sequences, the user invokes an operation on an object. For example, to delete a window, the user would select a window and invoke the delete operation. An extended keyboard permits the user to select many operations and standard items (e.g., window, image) at the touch of a key. The Item-Operation sequences are consistent and intuitive, allowing the user to infer new Item-Operation sequences from knowledge of previous sequences.

The user interface also provides an Ada command language with extensive command assistance capabilities, including command completion, stacking and retrieval, undo, prompting, semanticizing, and explanation (a form of help).

The help facility is extensive, context sensitive, and provides hierarchical levels of information on all objects of the Rational system including command keys and the help facility itself. A "What Does" function enables the user to get help on commands, and help menus present choices for getting further help. The environment also retains a help log for viewing previously retrieved help information.

Software Engineering Capabilities: As a language-centered environment, the critical capabilities of Rational are the tools that support code development (i.e., the Component Engineering phase of the SFLC).

Rational supports the Ada language with body stub generation, body template generation, pretty-printing, syntactic completion, semantic checking, and inline/separate unit conversions. Users are often unaware that the environment also includes a program library manager because it is so well integrated.

Rational also provides many capabilities for objects including comparison, searching, sorting, browsing, editing, creating, modifying, preserving, and deleting. Browsing capabilities include structured walkthroughs which enable users to view and traverse objects based on their definition and dependency relationships among other objects.

Rational compilation capabilities are significantly different from those the other environments provide. The Rational model is object-oriented. Ada units are single objects which pass through the following compilation phases: source, installed, and coded. Each phase has its own properties.

Rational provides incremental compilation capabilities which allow the user to make upwardly compatible changes to statements, comments, and declarations not referenced elsewhere in the program without requiring the recompilation of dependent units. The Rational also provides automatic recompilation of "withed" and dependent units for changes to units which are not upwardly compatible.

Rational provides multi-target support including target code generators and debuggers. Acting as a universal host, Rational provides a common interface to its target capabilities allowing the user to perceive them as a single host capability, not as target specific capabilities.

Reusability Support: Rational does not explicitly support reuse.

Other Support: Rational also provides those system management, configuration management, object management, and repository management capabilities that directly support code development.

Software Development and Maintenance Environment (SDME)

Background

The mission of the SDME is to "support WIS program personnel who develop and/or maintain WIS software products" (SDME85). WIS is the World Wide Military Command and Control System (WWMCCS) Information System. To fulfill this mission, the SDME has goals to support all users (e.g., technical, administrative, managerial) and all phases of the life cycle (e.g., development, maintenance).

The SDME is a four year (1985-1988) effort of the GTE Government Systems Division.

We used (SDME85), (SDME86), (SDME87), and (SDME88) in analyzing the SDME.

Key Capabilities

Operational Concept: The SDME is also a toolkit environment. The tool set is integrated from the user's perspective (i.e., command invocation) and internally (i.e., database access and updates). The SDME furnishes standard interfaces for integrating tools.

Foundation Capabilities: An object-oriented, Typed-Entity-Relationship-Attribute database supports strong type checking in the SDME. This database, a variation of the ERA model, requires a type attribute for most entities. The value of the type attribute defines which operations and properties are applicable to the entity.

The standard interfaces SDME provides for integrating tools are:

- the User Account Information Base Function
- the Command, Context and Type Information Base Function
- the Help Information Base Function
- the Reuse Information Base Function
- the Project Information Base Function
- the Project Management Information Base Function
- the Requirements Information Base Function
- the Tool Parameter Information Base Function.

The SDME also provides two functions that integrate host capabilities: the Virtual Core Functions and the Virtual Terminal Functions.

User Interface: The SDME has two interaction modes: a command line interface and a menu interface. The command language is a modified, Ada syntax language. User assistance aids provide command completion, aliasing, retrieval, and history retention. A full-screen menu provides selection lists of commands, types and objects, and the capability to build commands incrementally. Both interface access methods have help and tutorial capabilities.

Contexts, central to the SDME user interface, are a form of object and process control which restrict the user's focus. They narrow the scope of operations and objects available for processing based on the user's role (e.g., manager, programmer), access (e.g., read/write, group access, security level), and type or object selection. For example, if the user selects an object of type X, only the allowed operations for type X may be invoked in the current context.

Software Engineering Capabilities: The SDME provides typical program development capabilities including browsing, searching, compiling, and linking. A full-screen, template-driven text editor provides syntax assisted editing. Ada development support is minimal, but includes body stub generation, syntax checking, and cross reference capabilities.

The SDME also provides a minimal debugging facility and a few basic testing capabilities including test harness generation, coverage/frequency analysis, test case generation, and test data generation.

Reusability Support: While the SDME does not provide many reuse capabilities, it is the only implemented environment studied (the SSE intends to support reuse) to provide any reuse capability.

A reuse repository indexed by a hierarchical component catalog provides structure and information about the underlying components. Users may browse (code, specification and design), retrieve, modify, create, and insert components. They can also search for potential reuse candidates based on attributes. The repository, like a configuration manager, tracks component status and changes, component use, and dependencies on components. The catalog itself can also change: classifications can be added, deleted, reordered, and modified.

Other Capabilities: Complexity measurement, completeness checking, consistency checking, programming style checking, and cross reference analyzers support Quality Evaluation.

An extensive change control and tracking mechanism includes entry of problem reports/change requests (PRs/CRs), ownership authorization, PR/CR tracking, multiple solutions for a single PR/CR, and various reporting functions.

Space Station Software Support Environment (SSE)

Background

Lockheed Missiles and Space Company is developing the SSE for NASA. The SSE System Concept Document and System Functional Requirements Specifications, published in the first quarter of 1988, define capabilities on a higher level than the other environments studied. Unlike the other environments studied, the SSE is in development, therefore its capabilities are likely to change and evolve.

We used (SSE87), (SSE88a), and (SSE88b) in analyzing the SSE.

Key Capabilities

Operational Concept: The SSE is an "environment framework" environment.

Foundation Capabilities: Functionally, the SSE comprises three major elements: The SSE Development Facility (SSEDF), Software Productions Facilities (SPFs) and the Integration Facility (IF). The SSEDF, a superset of all "spawned" environments, is the base environment which controls the creation and servicing of operational elements. SPFs are environments tailored from the SSEDF for a particular software development project. SPFs are proper subsets of tools, methods, procedures, and reusable components from the base SSEDF. The IF provides system level testing and integration of all software.

The layered and interface driven SSE architecture comprises 4 major elements: the framework, tools, host system software, and a management facility including the SSE interfaces (Human-Computer, Distributed-Ada, and Test & Tool Harness).

All the elements of the SSE communicate via the Project Object Base. The Project Object Base is an integral part of the Database Management Element which provides users with a relational view and tools with a logical view of the underlying database. (No implementation details are available).

User Interface: A common user interface provides access to environment capabilities for users and provides standard interfaces for applications to communicate with users and other applications.

The SSE will have an "English-like" command language and help and training capabilities, but the SSE documents do not give other user interface details (e.g., interaction modes, command assistance).

Software Engineering Capabilities: The SSE will support the traditional program development capabilities such as editing, compiling, linking, and debugging. The SSE also intends to provide basic prototyping capabilities. Details are lacking for all program development capabilities.

Of the environments studied, the SSE is the only one to support modeling and simulation. In particular, it supports integration of models and simulations with product software, simulation of discrete, continuous, and network events, and real- and non-real-time execution of models and simulations.

The SSE provides extensive automated testing and integration management capabilities which range from test bed generation to regression test analysis. These capabilities include test execution, test response reporting, and product integration.

Software process management is key in the SSE. The interim SSE uses Automated Product Control Environment (APCE, a proprietary product) as its process manager. APCE is definition driven; for example, a project may define the life cycle, project organization, deliverables, and work breakdown structure. APCE uses these definitions in controlling the software development process.

Reusability Support: The SSE foundation supports reuse. Each time a project creates a SPF, it reuses and tailors the base SSEDF to its needs. During this process, the project assembles (from t'e base SSEDF) a repository of components for reuse by the project. During development, a project inserts components it develops into the base SSEDF repository for reuse by other projects. Detailed information about query capabilities and evaluation capabilities is unavailable.

Other Capabilities: The SSE documents focus on defining capabilities for configuration management, project management, and object management. The documents mention support for system management and repository management but do not detail capabilities.

Configuration management capabilities are extensive and include configuration control capabilities, system build capabilities, product release control, version control, and problem and change management.

The project management capabilities cover financial management, project definition, quality management, resource management, schedule management, and tracking more extensively than the other environments studied.

The SSE object management capabilities include all the capabilities found in the other environments plus consistency management (i.e., change information collection and change impact analysis).

Summary

The Environment Capability Matrix shows trends in software development support provided by the environments studied. Densely populated areas (a large number of "X"s) of the matrix represent capabilities which the environments support well. Large gaps (few or no "X"s) represent capabilities for which the environments provide little or no support.

The environments support traditional environment capabilities including system management, configuration management, project management, object management, library management, code development, static analysis, and operational maintenance capabilities. Generally, they do not support recently identified environment capabilities. In particular they do not support risk management, object dictionary management, project communication, and many phases of the SFLC (e.g., system architecture definition, system capability definition, prototype construction and evaluation, productization.)

Common and Unique Capabilities

Foundation Capabilities

Each environment studied uses a central database as its information repository and exchange medium. Details of the Rational and SSE databases are unavailable; the SDME and ALS/N use an ERA database.

In an ERA database, entities represent objects in the environment; relations represent an association between entities; and, attributes represent properties of entities. It is a powerful model for supporting environment capabilities. This data model supports configuration management, access control, and library management in the ALS/N and operational maintenance and contexts in the SDME.

The SSE environment framework supports environment growing: the tailoring of environments and repository assembly for project specific needs. Each time a project creates an environment, it reuses and tailors the base environment to its needs. During development a project inserts components, methods, and tools it develops into the base repository for reuse by other projects.

User Interface

Each environment provides a command line interface. Additionally, the SDME provides a menu interface and the Rational, SDME, and SSE provide windowing capabilities.

The capability to invoke Item-Operation sequences makes the Rational user interface object-oriented. The sequences provide a consistent and intuitive interface model for the user to communicate with the system.

Contexts focus the SDME user on the current task, eliminating all non-essential information from view. They narrow the scope of operations and objects available for processing based on the user's role (e.g., manager, programmer), access (e.g., read/write, group, security level), and type or object selection.

All the environments, except the SSE, provide an Ada, or an "Ada-like" command language. The SSE will provide an "English-like" command language. Command language assistance in the SDME and the Rational is extensive and includes command completion, aliasing, retrieval, history retention, and command building.

Each environment also provides a help facility. The Rational, SDME, and SSE facilities are context sensitive. Rational also provides a help log, hierarchical levels of information, and a "What Does" function that enables the user to determine what operations are available for a given topic. Both the SDME and SSE provide tutorial capabilities. The SSE capabilities are more extensive and include controlled command execution with user feedback. In addition, the SSE will provide capabilities to prepare computer aided and classroom instructional material.

Software Engineering Capabilities

Although the environments studied do not provide the exact same program development capabilities, each provides many of the same basic capabilities (e.g., editing, browsing, formatting, compiling, linking, debugging, searching, sorting, and comparing). Capabilities worth noting include:

- Editing
    - Syntax Sensitive (Rational, SDME, and SSE)
    - Semantic Completion (Rational)
- Browsing
    - Structured Walkthroughs (Rational)
- Compiling, Linking, Debugging
    - Object-Oriented (Rational)
    - Incremental Compilation (Rational)
    - Automatic Recompilation (Rational)
    - Multi-Target Support (ALS/N and Rational)

The SDME and Rational provide language assistance tools including body stub generation, body template generation, syntactic completion, semantic checking, and inline/separate unit conversions.

The SDME and the SSE provide testing support. The SDME testing support is minimal and includes test data generation, test harness generation, and coverage/frequency analysis. The SSE provides extensive automated testing support including test-bed generation, test execution, test response reporting, and regression test analysis.

Of the environments studied, the SSE is the only one to support modeling and simulation. In particular, it supports integration of models and simulations with product software, simulation of discrete, continuous, and network events, and real- and non-real-time execution of models and simulations.

Both the ALS/N and SSE provide software process management capabilities. The ALS/N command procedures provide a simple form of software process management. The SSE will provide more extensive capabilities and features a definition-driven process manager to control the software development process. Exact details arc unavailable.

Reusability Support

The SDME is the only environment studied to provide reuse capabilities although, the SSE intends to support reuse. It provides a hierarchical reuse repository and auxiliary capabilities including query, tracking, classification maintenance, and component browsing, retrieval, modification, and insertion.

Reusability support is central to the SSE: its environment framework relies on reuse capabilities. Each time a project creates an environment, it reuses and tailors the base environment to its needs. During this process, the project assembles (from the base environment) a repository of components for reuse. During development, the project inserts components it develops into the base repository for reuse by other projects.

Recommendations

Recommendations for SEE Capabilities

We recommend the following software engineering capabilities based on our analyses.

- Integration

  "The essence of a software environment is the synergistic integration of tools in order to provide strong, close support for a software job (Osterw81)." The Rational is an excellent example of a highly integrated software environment. The SEE needs to broaden Rational's scope of software engineering capabilities while maintaining a level of integration at least as good as that of Rational.

- Environment Framework

  The SSE environment framework is an excellent high level functional design for the SEE. It supports environment growing: the tailoring of environments and repository assembly for project specific needs. Environment frameworks are extensible and provide an adaptable base for environment support.

- Use of a Central Database

  (Osterw81) cites a central database as one of the most important characteristics of a well-integrated software environment. Each environment studied uses a central database as its information repository and exchange medium.

  The ALS/N uses the ERA model to provide well-integrated program library management, configuration management and system management capabilities.

- Adaptable Software Process Management

  Hardwiring management support for particular software process is contrary to the STARS Program goal of adaptability to change. The SSE provides an example of an adaptable, definition-driven software process manager. It supports project specific processes and methods well. The SEE should develop a similar software process manager.

- Software Engineering

  - Ada Development Support

    Overall, the development tools of the Rational provide a good model for the SSE. Its capabilities include well-integrated editing, browsing, searching, sorting, compiling, linking, and debugging.

    ▲ Well-Integrated Library Manager

    The SEE must include a well-integrated program library manager to give both programmers and other tools access to program library information. Both ALS/N and Rational provide examples of well-integrated library managers.

    ▲ Ada Compilation

    Key capabilities for Ada compilation in the SEE must include:

            - △ Incremental Compilation (Rational)
            - △ Automatic Recompilation (Rational)
            - △ Multi-Target Support (ALS/N and Rational)

        - ▲ Ada Support Capabilities

          The SEE must incorporate tools which make the Ada language easier to use. Rational provides many of these: body stub generation, body template generation, syntactic completion, semantic checking, structured walkthroughs, and inline/separate unit conversions

        - ▲ Multi-Target Support

          Portability goals require the SEE to provide multi-target support. ALS/N, one of the environments providing multi-target support, includes a multi-target compiler, and target linkers, debuggers, and profilers.

    - ▪ Testing and Integration

      The testing and integration capabilities of the SSE automate testing functions previously done manually. The SEE must provide comparable automating capabilities.

    - ▪ Prototyping, Simulation, and Modeling

      Prototyping, simulation, and modelling capabilities are all necessary to support the SFLC. If the SFLC is to be the primary life cycle model of the SEE, the SEE must provide these capabilities.

- Human Computer Interfaces

    - ▪ User Roles

      Contexts, provided by the SDME user interface, focus the user on the current task, eliminating all non-essential information from view. While somewhat constraining, contexts provide a level of abstraction necessary for user to function in different roles. As environments (including the SEE) grow in function/capabilities, and as users take on multiple roles this abstraction capability will be necessary for the user to work effectively.

    - ▪ User Display Interface

      The Rational's well integrated user display interface is a good model for the SSE. Item-Operation sequences make the Rational user interface object-oriented and present an intuitive and consistent interface model to the user. The interface also provides an Ada command language with extensive command assistance capabilities and an extensive, easily retrievable context sensitive help facility.

- Standard Internal Interfaces

  Standard internal interfaces, critical to a well-integrated environment, ensure commonality, portability, and standardization. The SSE must provide interfaces similar to the Rational and SSE models. For the SEE to be well-integrated, it must provide standard internal interfaces. Both the Rational and SSE interface capabilities are good models to analyze further.

- Reuse

  Reuse is a major goal of the STARS Program that the SEE must support. While it is almost certain a reuse repository will be necessary, it is too early in the study of reuse to recommend many detailed capabilities. Instead, we recommend studying the Q phase reuse results to refine reuse requirements/capabilities.

Other Recommendations

We recommend focusing on the environment framework and integration mechanisms before placing emphasis on environment tools. We feel that the framework is the basis of the environment and that a good framework will extend to incorporate well-integrated tools.

We also recommend prototyping efforts to refine fuzzy capabilities (e.g., reuse). This includes a recommendation for studying the Software First Life Cycle thoroughly.

Lessons Learned

- To comprehend the hundreds of environment capabilities, we found it necessary to organize them in some fashion. We felt the simplest method was to create a taxonomy.
- We found that a hierarchical taxonomy cannot fully represent an iterative or recursive life cycle. It fully represents the capabilities that make up the life cycle, but cannot capture the essence of control and data flows through the life cycle.
- A taxonomy based on capabilities cannot fully represent attributes like "tailorable" and "written in Ada" and consequently is of little use in making qualitative assessments of environments.
- While reuse is easy to imagine, it is sometimes difficult to achieve. In retrospect, it might have been more time efficient to draft a new environment capability taxonomy than to start with the NBS Taxonomy. Initially, reusing the NBS Taxonomy seemed like a good idea, but its deficiencies concerning the SFLC made it less than a perfect candidate for reuse. To achieve the benefits of reuse, the effort to modify an object must be less than the effort to develop it from scratch.
- Even though the NBS Taxonomy turned out to be ill-suited for reuse in this application, it was valuable in the same way that the initial prototype is valuable in the SFLC. We (being our own customers) did not know what we wanted our final taxonomy to resemble. We started with a concept, a taxonomy, but had little idea that the NBS Taxonomy was not what we wanted. Once we established the NBS Taxonomy as our initial prototype and started to evaluate it, we easily recognized areas which needed improvement. The first prototype was necessary to show us the direction not to proceed.
- To reuse documents, they must be in soft copy and retrievable in a format consistent with (or malleable to) that of the document in production. Our copy of the NBS Taxonomy was a paper copy not suitable for optical scanning. Retyping text was inefficient use of our limited resources.
- We did not understand the SFLC well when we started, making it difficult to classify environment capabilities. From this study, we know more about the SFLC, but on the whole, we still know little about it. It will take several software developments using the SFLC for us to refine and become comfortable with our definition.

Acronyms

Acronym	*Meaning*
ALS/N	Ada Language System/Navy
APSE	Ada Programming Support Environment
COTS	Commercial off the Shelf (Software)
CDRL	Contract Data Requirements List
CR	Change Request
DIANA	Descriptive Intermediate Attributed Notation for Ada
DoD	(United States) Department of Defense
ERA	Entity-Relationship-Attribute
IBM	International Business Machines
IF	(SSE) Integration Facility
MAPSE	Minimal Ada Programming Support Environment
NASA	National Aeronautics and Space Administration
NBS	National Bureau of Standards
PR	Problem Report
SEE	Software Engineering Environment
SDME	Software Development and Maintenance Environment
SFLC	Software First Life Cycle
SOW	Statement of Work
SPF	(SSE) Software Productions Facility
SSEDF	SSE Development Facility
STARS	Software Technology for Adaptable, Reliable Systems
SSE	(NASA) Space Station Software Support Environment
TCSEC	Trusted Computer System Evaluation Criteria
UI	User Interface
WIS	WWMCCS Information System
WWMCCS	World Wide Military Command and Control System

References

(ALS/N86) Naval Ocean Systems Center, *System Specification for Ada Language System/Navy*, NAVSEA 0967-LP-598-9710, October 8, 1986.

(Chen76) Chen, P.P., "The Entity Relationship Model: Toward a Unified View of Data," *ACM Transactions on Data Base Systems*, Vol. 1, No. 1, March 1976.

(Dart87) Dart, Susan A., et al., "Software Development Environments", *IEEE Computer*, October 1987.

(DoD2167) United States Department of Defense, *Defense System Software Development*, DoD-STD-2167, June 4, 1985.

(E&VCS87) *E & V Classification Schema Report*, AFWAL/AAAF, Wright-Patterson AFB, Ohio, Version 1.0, June 15, 1987.

(E&VRM88) *E & V Reference Manual*, TASC Report No. TR-5234-3, Version 1.0, AFWAL/AAAF, Wright-Patterson AFB, Ohio, March 15, 1988.

(Goos83) Goos, G., and Hartmanis, J., *DIANA: Descriptive Intermediate Attributed Notation for Ada*, Springer-Verlag, 1983.

(Hough82) Houghton, Raymond C., *A Taxonomy of Tool Features for the Ada Programming Support Environment (APSE)*, National Bureau of Standards, Draft of August 9, 1982.

(IBM0100) IBM Systems Integration Division, *Environment Capability Matrix*, CDRL Sequence No. 0100, December 17, 1988.

(IBM0090) IBM Systems Integration Division, *Environment Requirements Report*, CDRL Sequence No. 0090, March 17, 1989.

(Kean85) Kean, Elizabeth S., and LaMonica, Frank S., *A Taxonomy of Tool Features for a Life Cycle Software Engineering Environment*, Rome Air Development Center In-House Report No. RADC-TR-85-112, June 1985.

(Lyons86) Lyons, T.G., and Nissen, J.D., *Selecting an Ada Environment*, Cambridge University Press, Cambridge, England, 1986.

(Osterw81) Osterweil, L. J., "Software Environment Research: Directions For the Next Five Years", *Computer*, Vol. 14, No. 4, April 1981.

(Pened88) Penedo, Maria H., and Riddle, William E., *Software Engineering Environment Architectures*, Guest Editors' Introduction, IEEE Transactions on Software Engineering, Vol 14., No. 6, June 1988.

(Ratio85) Rational, *Products and Technology*, 1885.

(Ratio87) Rational, *Rational R1000 Development System Reference Manuals*, 1987.

(Ratio88) *Rational Design Facility: DOD-STD-2167, Technical Specification*, Document Number TS-4, Rev. 1.0, June 1988.

(SDME85) GTE Government Systems, *Draft Functional Design Specification for the Software Development and Maintenance Environment*, WIS-SPEC-801, June 21, 1985.

(SDME86) GTE Government Systems, *Formal Draft System Specification for the Software Development and Maintenance Environment*, Control No. 1302-2, June 30, 1986.

(SDME87) GTE Government Systems, *Software User's Manual for the Software Development and Maintenance Environment Working Prototype System*, February 5, 1987.

(SDME88) GTE Government Systems, *Draft Software Requirements Specification for the Software Development and Maintenance Environment*, WIS-SPEC-803, April 27, 1988.

(SSE87) Lockhead Missiles and Space Company, *Interim SSE System User's Guide*, Space Station Software Support Environment, LMSC F255423, Rev. 1.0, December 14, 1987.

(SSE88a) Lockheed Missiles and Space Company, *System Concept Document*, Space Station Software Support Environment, LMSC F255415, Version 1.0, January 15, 1988.

(SSE88b) Lockheed Missiles and Space Company, *System Functional Requirements Specification*, Space Station Software Support Environment, LMSC F255416, March 10, 1988.

(Sterl88) Sterlich, T., "The Software Life Cycle Support Environment (SLSCE) a Computer Based Framework for Developing Software Systems", *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development*, Boston, Massachusetts, November 28-30, 1988. Available in "Software Engineering Notes" *SIGSOFT*, Vol. 23, No. 5, November 1988.

(STONE80) United States Department of Defense, *Requirements for Ada Programming Support Environments*, "STONEMAN," February 1980.

(TCSEC83) United States Department of Defense, *Trusted Computer System Evaluation Criteria (TCSEC)*, CSC-STD-001-83.

(Weide87) Weiderman, Nelson, et al., *Evaluation of Ada Environments*, Carnegie-Mellon University, Software Engineering Institute, Technical Report No. CMU/SEI-87-TR-1, March 1987.

Appendix A. Environment Capability Matrix

The following are notes concerning the matrix which begins on the next page. NOTES:

- The taxonomy should not be construed as a list of requirements for environments, but rather as a tool for exploring environment capabilities.
- Since our analysis is based on documentation, capabilities may exist in the implemented environments although we have not shown them in our matrix.
- The variance between environment documents also makes it difficult to uniformly evaluate the environments. For example, the SSE specifies it will provide debugging support but provides no details. In contrast, Rational lists extensive debugging capabilities.
- The SEE capabilities are included in the matrix for reference and comparison.
- Definitions for the terms used in the taxonomy/matrix are given in section "Appendix B. Capability Definitions" on page 29.
- We eliminated identical lists of capabilities from all but one location in the taxonomy. We note removal of capabilities with a marker like the following: (see 3.2.2.1 Capability Foo).
- To help distinguish the phases of the SFLC, we labelled them with a with a trailing "(*)."

Environments					
STARS SEE	ALS/N	RATIONAL	SDME	SSE	Capabilities
X	X	X	X	X	1 Management Capabilities
	X	X	X	X	1.1 Computer System Management
	X	X	X	X	1.1.1 Access Management
X	X	X	X	X	1.1.1.1 Access Control
	X	X	X	X	1.1.1.1.1 Discretionary Control
	X	X	X	X	1.1.1.1.2 Group Definition
	X				1.1.1.1.3 Mandatory Control
	X		X	X	1.1.1.1.4 Object Control
			X	X	1.1.1.1.5 Process Control
	X	X	X	X	1.1.2 System Resource Management
	X			X	1.1.2.1 Accounting Information Collection
	X	X	X	X	1.1.2.2 Archival, Backup, and Recovery
				X	1.1.2.3 Configuration Modification
X				X	1.1.2.4 Workload Monitoring
	X	X	X		1.1.3 User Account Management
	X		X		1.1.3.1 Initial Working Directory Creation
	X				1.1.3.2 Login/Logout Macro Creation
	X	X	X		1.1.3.3 User Account Group Maintenance
	X	X	X		1.1.3.4 User Account Maintenance
X	X	X	X	X	1.2 Configuration Management
X	X	X	X	X	1.2.1 Configuration Control
X	X	X	X	X	1.2.1.1 Baseline Definition
X	X			X	1.2.1.2 Foreign Object Import
X	X	X		X	1.2.1.3 Multiple Development Path Creation
	X	X			1.2.1.4 Source Code Support
	X	X			1.2.1.4.1 Alternate Implementation Specification
		X			1.2.1.4.2 Meta-Package/Subsystem Support
X	X	X	X	X	1.2.1.5 System Build Definition
X	X	X		X	1.2.1.5.1 Object Dependency Specification
X		X		X	1.2.1.5.2 Object Specification
	X				1.2.1.5.3 Translator Specification
X	X	X	X	X	1.2.1.6 System Build
				X	1.2.1.6.1 Current Default Build
				X	1.2.1.6.2 History Capture
	X			X	1.2.1.6.3 Hybrid Build
		X		X	1.2.1.6.4 Inconsistent System Warning
					1.2.1.6.5 Partial/Incomplete Build
	X				1.2.1.6.6 Specific Change or Problem Build
		X	X	X	1.2.1.6.7 Specific Release Build (Rebuild)
X	X	X	X	X	1.2.2 Product Release Control
X	X	X	X	X	1.2.2.1 Release Generation
			X	X	1.2.2.2 Release History Capture
			X	X	1.2.2.2.1 Differences Among Releases
			X	X	1.2.2.2.2 Errors Reported/Fixed by Release
					1.2.2.2.3 Hardware Configuration for Release
				X	1.2.2.2.4 Locations of Release
			X	X	1.2.2.2.5 Release Identification
					1.2.2.2.6 Software Configuration for Release
X	X	X	X	X	1.2.3 Version Control
	X			X	1.2.3.1 Comparison
	X	X		X	1.2.3.2 Creation
	X	X		X	1.2.3.2.1 Initial
	X	X		X	1.2.3.2.2 Successor
	X			X	1.2.3.2.3 Variant
	X	X		X	1.2.3.3 History Capture
		X			1.2.3.4 Merging
X	X	X	X	X	1.2.3.5 Reservation

		X		X	1.2.3.3 Retrieval
					1.2.3.3.1 Explicit
	X				1.2.3.3.2 Dynamic
					1.2.3.3.3 Referential
X			X	X	1.2.4 Problem and Change Management
X			X	X	1.2.4.1 Assignment and Authorization
					1.2.4.2 Association with Test Cases
X				X	1.2.4.3 Cataloging
X			X	X	1.2.4.4 Report/Status Generation
			X	X	1.2.4.5 Resolution Text Capture
	X			X	1.3 Software Process Management
				X	1.3.1 Software Process Definition
					1.3.1.1 Software Process Type Definition
					1.3.1.2 Software Process Instance Creation
					1.3.1.3 Invocation Condition Specification
	X			X	1.3.2 Process Sequencing
					1.3.3 Consistency Checking and Updating
X			X	X	1.4 Project Management
X			X	X	1.4.1 Financial Management
X				X	1.4.1.1 Cost Actuals Collection
X			X	X	1.4.1.2 Cost Assessment
X			X		1.4.1.3 Cost Estimation
X			X	X	1.4.1.4 Work Breakdown Structure
					1.4.2 Project Definition
				X	1.4.2.1 Organization Definition
X		X	X	X	1.4.2.1.1 Responsibility Assignment
			X	X	1.4.2.2 Project Initiation
			X		1.4.2.3 Project Deletion
X			X	X	1.4.3 Quality Management
X			X	X	1.4.3.1 Auditing
X			X	X	1.4.3.2 Evaluating
X				X	1.4.3.3 Project Procedures Definition
X			X	X	1.4.3.4 Project Standards Definition
X			X	X	1.4.3.5 Quality Factor Definition
X			X		1.4.3.6 Quality Prediction
X			X	X	1.4.4 Resource Management
				X	1.4.4.1 Assessment
					1.4.4.2 Balancing
X			X	X	1.4.4.3 Definition
X			X	X	1.4.4.4 Estimation
					1.4.5 Risk Management
					1.4.5.1 Object to Risk Category Association
					1.4.5.2 Risk Category Definition
					1.4.5.3 Risk Management Text Definition
					1.4.5.3.1 Category
					1.4.5.3.2 Object
					1.4.5.4 Status Reporting
			X	X	1.4.6 Schedule Management
			X		1.4.6.1 Baseline Maintenance
X					1.4.6.2 Calendar Management
X				X	1.4.6.2.1 Individual
X			X		1.4.6.2.2 Project
					1.4.6.3 Contingency Analysis
X			X	X	1.4.6.4 Critical Path Definition
			X		1.4.6.5 Earned Value Calculation
X			X		1.4.6.6 Milestone Definition
X			X		1.4.6.7 Schedule Assessment
			X		1.4.6.8 Schedule Updating and Recalculation
X				X	1.4.7 Tracking
X				X	1.4.7.1 Cost
X				X	1.4.7.2 Project Data
X				X	1.4.7.3 Quality
X				X	1.4.7.4 Resource Use
X				X	1.4.7.5 Technical Performance
X		X	X	X	1.5 Project Repository Management
X			X		1.5.1 Repository Administration
			X		1.5.1.1 Concurrent Access
			X		1.5.1.2 Maintenance
					1.5.1.3 Navigation Mechanism Definition
					1.5.1.3.1 Hierarchical

					1.5.1.3.2 Networked
					1.5.1.4 Query Language
X			X	X	1.5.1.5 Reuse Repository Management
				X	1.5.1.5.1 Acceptance Criteria Establishment
X				X	1.5.1.5.2 Cataloging
X				X	1.5.1.5.3 Search/Retrieval
	X			X	1.5.2 Documentation Management
X				X	1.5.2.1 Document Template Definition
					1.5.2.2 Document Type Definition
				X	1.5.2.3 Integrated Text and Graphics
				X	1.5.2.3.1 User Interface Display Capture
X				X	1.5.2.4 Processing
X				X	1.5.2.5 Rerelease Analysis
					1.5.2.5.1 Change Bars
					1.5.2.5.2 Page Renumbering/Insertion
	X		X	X	1.5.2.6 Source Code Publication
	X		X	X	1.5.2.6.1 Formatting
	X		X	X	1.5.2.6.2 Indexing/Headers/Table of Contents
	X				1.5.2.6.3 Macro Processing
					1.5.2.6.4 Pragma Sensitivity
	X	X	X	X	1.5.3 Object Management
X				X	1.5.3.1 Consistency Management
X				X	1.5.3.1.1 Analysis
					1.5.3.1.2 Definition
X				X	1.5.3.1.3 Change Impact Analysis
X				X	1.5.3.1.4 Change Information Collection
				X	1.5.3.1.5 Reporting
					1.5.3.2 Maintenance
	X	X	X	X	1.5.3.2.1 Creation
	X		X	X	1.5.3.2.1.1 Attribute Definition
	X		X	X	1.5.3.2.1.2 Relationship Definition
			X		1.5.3.2.1.2.1 Allowed Processes Definition
X					1.5.3.2.1.2.2 Standards Processing Definition
	X	X	X	X	1.5.3.2.2 Deletion
	X	X	X	X	1.5.3.2.3 Modification
X					1.5.3.3 Object Dictionary Management
X					1.5.3.3.1 Cross Reference
					1.5.3.3.2 Interactive Query
					1.5.3.3.3 Navigation
					1.5.3.3.4 Object and Relationship Display
					1.5.3.3.5 Reporting
					1.5.3.4 Owner Assignment
X	X	X	X	X	1.5.3.5 Program Library (Ada) Management
	X	X	X		1.5.3.5.1 Creation/Deletion/Maintenance
	X			X	1.5.3.5.2 Foreign Code Import
		X			1.5.3.5.3 Navigation
	X	X			1.5.3.5.4 Program Library Relationship Specification
	X				1.5.3.5.5 Query
	X				1.5.3.5.5.1 Completeness/Recompilation Status
	X				1.5.3.5.5.2 Relationships
	X				1.5.3.5.5.3 Unit Information
	X	X			1.5.3.6 Retrieval
X	X	X	X	X	2 Development Engineering Capabilities
					2.1 Concept Development (*)
X				X	2.2 Environment Growing (*)
X				X	2.2.1 Environment Tailoring (*)
				X	2.2.2 Repository Assembly (*)
					2.3 System Development (*)
					2.3.1 System Architecture Definition (*)
					2.3.1.1 Software Architecture Definition (*)
					2.3.1.1.1 Interface Selection/Definition (*)
X					2.3.1.1.1.1 Architecture Rationale Capture
					2.3.1.1.1.2 Control Flow Definition
					2.3.1.1.1.3 Data Flow Definition
					2.3.1.1.1.4 Object and Operation Definition
					2.3.1.1.1.5 Program Unit Interface Definition
X					2.3.1.1.1.5.1 Graphical System Description
					2.3.1.1.1.5.2 System Description to Ada Conversion
					2.3.1.1.2 Interface/Architecture Evaluation (*)

					2.3.1.2 Hardware Architecture Definition (*)
X					2.3.1.2.1 Resource/Trade-off Studies (*)
X				X	2.3.1.2.1.1 Simulation and Modelling
					2.3.1.2.2 Hardware Definitization (*)
					2.3.1.2.3 Hardware Selection (*)
					2.3.2 System Capability Definition (*)
					2.3.2.1 Preliminary Capability Statement (*)
					2.3.2.2 Capability Definitization (*)
					2.3.2.3 Formal Capability Statement (*)
					2.3.3 Prototype Competition (*)
			X		2.3.3.1 Alternative Formulation (*)
X					2.3.3.1.1 Alternative Consideration (*)
					2.3.3.1.2 Breakthrough Consideration (*)
X			X	X	2.3.3.1.3 Alternative Evaluation (*)
X				X	2.3.3.1.3.1 Evaluation Results Capture
X				X	2.3.3.1.3.2 Selection Rationale Capture
				X	2.3.3.2 Prototyping (*)
					2.3.3.2.1 Component Acquisition (*)
X			X		2.3.3.2.1.1 Component Reuse Analysis (*)
					2.3.3.2.1.1.1 Component Selection Guidance
				X	2.3.3.2.1.2 Component Development/Refinement (*)
X				X	2.3.3.2.1.2.1 Component Prototyping (*)
					(see 2.3.3.2 Prototyping (recursive))
	X	X	X	X	2.3.3.2.1.2.2 Component Engineering (*)
		X			2.3.3.2.1.2.2.1 Specification Development (*)
X					2.3.3.2.1.2.2.1.1 Component Development Folder
		X	X		2.3.3.2.1.2.2.1.2 Object and Operation Definition/Redefinition
		X			2.3.3.2.1.2.2.1.3 Program Unit Interface Definition/Redefinition
					2.3.3.2.1.2.2.1.4 Static Analysis
					2.3.3.2.1.2.2.1.4.1 Type Analysis
					2.3.3.2.1.2.2.1.4.2 Unit Analysis
					2.3.3.2.1.2.2.1.5 Translation
					(see 2.3.3.2.1.2.2.2.2 Translation)
	X	X	X	X	2.3.3.2.1.2.2.2 Implementation Development (*)
		X			2.3.3.2.1.2.2.2.1 Body Development Support
X	X	X	X		2.3.3.2.1.2.2.2.1.1 Body Stub Generation
		X			2.3.3.2.1.2.2.2.1.2 Body Template Generation
					2.3.3.2.1.2.2.2.1.3 Commentary Transfer
		X			2.3.3.2.1.2.2.2.1.4 Inline-Separate Unit Conversion
X	X	X	X	X	2.3.3.2.1.2.2.2.2 Translation
X	X		X	X	2.3.3.2.1.2.2.2.2.1 Assembling
X					2.3.3.2.1.2.2.2.2.1.1 Macro Expansion
X					2.3.3.2.1.2.2.2.2.1.2 Structure Preprocessing
X	X	X	X	X	2.3.3.2.1.2.2.2.2.2 Compilation
					2.3.3.2.1.2.2.2.2.2.1 Cross
		X			2.3.3.2.1.2.2.2.2.2.2 Incremental
	X	X			2.3.3.2.1.2.2.2.2.2.3 Intermediate Language Generation
		X			2.3.3.2.1.2.2.2.2.2.4 Recompilation of Withed and Dependent Units
	X	X			2.3.3.2.1.2.2.2.2.2.5 Semantic Checking
	X	X	X		2.3.3.2.1.2.2.2.2.2.6 Syntax Checking
X					2.3.3.2.1.2.2.2.2.3 Conversion
X					2.3.3.2.1.2.2.2.2.4 Decompilation
X					2.3.3.2.1.2.2.2.2.5 Disassembly
X					2.3.3.2.1.2.2.2.2.6 Source Reconstruction
X	X	X	X	X	2.3.3.2.1.2.2.2.3 Debugging
	X	X			2.3.3.2.1.2.2.2.3.1 Cross-Debugging
X					2.3.3.2.1.2.2.2.3.1.1 Target Machine Instruction Simulation
					2.3.3.2.1.2.2.2.3.2 Machine Level Debugging
					2.3.3.2.1.2.2.2.3.3 Simultaneous Symbolic and Machine Level Debugging
X	X	X	X	X	2.3.3.2.1.2.2.2.3.4 Symbolic Debugging
	X	X	X		2.3.3.2.1.2.2.2.3.4.1 Breakpoint Setting/Clearing
	X	X	X		2.3.3.2.1.2.2.2.3.4.2 Execution Control
	X	X	X		2.3.3.2.1.2.2.2.3.4.3 Program State Modification
	X	X	X		2.3.3.2.1.2.2.2.3.4.4 Program State Query
X		X			2.3.3.2.1.2.2.2.3.5 Memory Dump
X	X	X	X	X	2.3.3.2.1.2.2.2.4 Linking/Loading
	X	X			2.3.3.2.1.2.2.2.4.1 Cross-Linking
	X	X			2.3.3.2.1.2.2.2.4.2 Partial Linking
X			X		2.3.3.2.1.2.2.3 Component Testing (*)
X			X	X	2.3.3.2.1.2.2.3.1 Coverage/Frequency Analysis

X					2.3.3.2.1.2.2.3.2 Mutant Analysis
X					2.3.3.2.1.2.2.3.3 Mutant Generation
X					2.3.3.2.1.2.2.3.4 Target Support
X					2.3.3.2.1.2.2.3.4.1 Host-Target Upload-Download
					2.3.3.2.1.2.2.3.4.1.1 Data
					2.3.3.2.1.2.2.3.4.1.2 Executable Image
					2.3.3.2.1.2.2.3.4.1.3 Object Code
					2.3.3.2.1.2.2.3.4.1.4 Source Code
X					2.3.3.2.1.2.2.3.4.2 Target Code Generation
X			X	X	2.3.3.2.1.2.2.3.5 Test Case/Data Generation
			X		2.3.3.2.1.2.2.3.5.1 Boundary Case Testing
X					2.3.3.2.1.2.2.3.5.2 Expected Output Generation
					2.3.3.2.1.2.2.3.5.3 Functional Testing
					2.3.3.2.1.2.2.3.5.4 Stress Testing
					2.3.3.2.1.2.2.3.5.5 Structural Testing
X				X	2.3.3.2.1.2.2.3.5.6 Test Case Specification
X					2.3.3.2.1.2.2.3.6 Test Execution
X				X	2.3.3.2.1.2.2.3.6.1 Initial Testing
X				X	2.3.3.2.1.2.2.3.6.2 Regression Analysis
X				X	2.3.3.2.1.2.2.3.6.3 Regression Testing
X			X	X	2.3.3.2.1.2.2.3.7 Test Harness Generation
X					2.3.3.2.1.2.2.3.8 Test Result Management
X					2.3.3.2.1.2.2.3.8.1 Actual and Expected Data Comparison
X					2.3.3.2.1.2.2.3.8.2 Data Analysis
X					2.3.3.2.1.2.2.3.8.3 Data Reduction
X				X	2.3.3.2.1.2.2.3.8.4 Logging
X				X	2.3.3.2.1.2.2.3.8.4.1 Configuration
X					2.3.3.2.1.2.2.3.8.4.2 History
X				X	2.3.3.2.1.2.2.3.8.4.3 Result
X				X	2.3.3.2.1.2.2.3.8.5 Result Reporting
					(see 2.3.3.2.4.1 Component Quality Evaluation)
					2.3.3.2.2 Prototype Construction (*)
					2.3.3.2.2.1 Framework Specification (*)
					(see 2.3.3.2.1.2.2.1 Specification Development)
					2.3.3.2.2.2 Framework Implementation and Component Hookup (*)
					(see 2.3.3.2.1.2.2.2 Implementation Development)
					2.3.3.2.2.3 Prototype Exercising and Informal Testing (*)
					(see 2.3.3.2.1.2.2.2.3 Debugging)
					(see 2.3.3.2.1.2.2.3 Component Testing)
					2.3.3.2.3 Prototype Evaluation (*)
					2.3.3.2.3.1 User Interface Analysis
					2.3.3.2.3.1.1 User Action Tracking
					2.3.3.2.3.1.2 User Error Tracking
					2.3.3.2.3.1.3 User Feedback Capture
					2.3.3.2.3.1.4 Real-Time UI Modification and Resumption
					2.3.3.2.4 Component Productization (*)
X			X	X	2.3.3.2.4.1 Component Quality Evaluation (*)
	X	X	X	X	2.3.3.2.4.1.1 Static Analysis
			X	X	2.3.3.2.4.1.1.1 Complexity Measurement
X			X		2.3.3.2.4.1.1.2 Completeness Checking
X			X	X	2.3.3.2.4.1.1.3 Consistency Checking
X		X	X		2.3.3.2.4.1.1.4 Cross Reference
X	X	X	X		2.3.3.2.4.1.1.4.1 Reference Only
X					2.3.3.2.4.1.1.4.2 Uninitialized Variable
X					2.3.3.2.4.1.1.4.3 Unused Variable
					2.3.3.2.4.1.1.4.4 Value Change
X					2.3.3.2.4.1.1.5 Data Flow Analysis
					2.3.3.2.4.1.1.6 Exception Analysis
	X		X	X	2.3.3.2.4.1.1.7 Programming Style Analysis
					2.3.3.2.4.1.1.8 Statement Profiling and Analysis
					2.3.3.2.4.1.1.9 Structure Checking
				X	2.3.3.2.4.1.2 Dynamic/Performance Analysis
				X	2.3.3.2.4.1.2.1 Resource Utilization
X				X	2.3.3.2.4.1.2.2 Timing
					2.3.3.2.4.1.2.2.1 Subprogram
X	X	X			2.3.3.2.4.1.2.3 Tracing
					2.3.3.2.4.1.2.3.1 Data Flow Tracing
X	X	X			2.3.3.2.4.1.2.3.2 Path Flow Tracing
	X				2.3.3.2.4.1.2.3.3 Tracepoint Activation/Deactivation
					2.3.3.2.4.1.2.3.4 Tracepoint Display
					2.3.3.2.4.2 Formal Component Testing (*)
					(see 2.3.3.2.1.2.2.3 Component Testing)
					2.3.3.2.4.3 Component Refinement (*)

X					2.3.3.2.4.3.1 Optimization
X					2.3.3.2.4.3.2 Preamble Generation
					2.3.3.2.4.3.3 Tuning
X			X	X	2.3.3.2.4.4 Repository Insertion (*)
					2.3.3.3 Prototype Selection (*)
					2.4 Productization (*)
				X	2.4.1 System Quality Evaluation (*)
					(see 2.3.3.2.4.1 Component Quality Evaluation)
					(see 2.5.4 Reliability Analysis)
				X	2.4.2 System Testing (*)
					(see 2.3.3.2.1.2.2.3 Component Testing)
					2.4.3 System Refinement (*)
					2.4.3.1 Optimization
					(see 2.3.3.2.4.3.1 Optimization)
					2.4.4 Final User and System Documentation Compilation (*)
X			X	X	2.5 Operational Maintenance (*)
X			X	X	2.5.1 Problem Reporting
			X	X	2.5.1.1 Problem Report Analysis
X				X	2.5.1.2 Impact Analysis
X			X	X	2.5.2 Change Requesting
			X	X	2.5.2.1 Change Request Analysis
X				X	2.5.2.2 Impact Analysis
				X	2.5.3 Performance Analysis
					(see 2.3.3.2.4.1.2 Dynamic/Performance Analysis)
X			X		2.5.4 Reliability Analysis
					2.5.5 Software Updating
					2.5.5.1 Patching Software
					2.5.5.2 Replacing Software
X	X	X	X	X	3 Communication Capabilities
X	X	X	X	X	3.1 Help Facility
X		X	X	X	3.1.1 Context-Sensitive
		X			3.1.3 Log
X	X	X	X	X	3.1.3 On-line Documentation
X	X			X	3.1.4 Tutorial/Training
X				X	3.1.4.1 Controlled Command Execution
X					3.1.4.2 Tutorial History Capture
X					3.1.4.3 User Learning Evaluation
X					3.2 Project Communication
					3.2.1 Demonstration Chart Preparation
X					3.2.2 Electronic Conferencing
					3.2.3 Electronic Forum
X				X	3.2.4 Electronic Mail
X	X	X	X	X	3.3 User Interface
X	X	X	X		3.3.1 Command Support
			X		3.3.1.1 Aliasing
		X	X		3.3.1.2 Building
		X	X		3.3.1.3 Completion
			X		3.3.1.4 Contexts
		X	X		3.3.1.5 Confirmation of Destructive Action
		X			3.3.1.6 Error Recovery
		X	X		3.3.1.7 Feedback
		X	X		3.3.1.8 History Log
	X	X	X		3.3.1.9 Procedures
		X	X		3.3.1.10 Stacking and Retrieval
		X		X	3.3.1.11 Undo
	X	X	X	X	3.3.2 Interaction Modes
X		X	X		3.3.2.1 Command Line
			X		3.3.2.2 Graphical
		X	X		3.3.2.3 Menu
		X	X		3.3.2.4 Object-Oriented
					3.3.2.5 Pointing Device
		X	X	X	3.3.2.6 Windowing

X	X	X	X	X	4 Global Object Manipulation Capabilities
	X	X	X	X	4.1 Browsing
					4.1.1 Structured Walkthrough
	X	X	X		4.2 Comparison
X	X	X	X	X	4.3 Editing
X			X		4.3.1 Graphics
X	X	X	X		4.3.2 Text
X	X	X	X		4.3.2.1 Screen
X	X				4.3.2.2 Line
		X			4.3.2.3 Semantic Completion
X		X	X	X	4.3.2.4 Syntax Assisted
	X	X	X		4.4 Formatting
X					4.5 Output
X					4.5.1 Graphics
					4.5.1.1 File
X					4.5.1.2 Hardcopy
X					4.5.1.3 Screen
X					4.5.2 Graphics Interchange Format
X					4.5.3 Object Interchange Format
X		X	X		4.5.4 Text
					(see 4.5.1 Graphics)
		X	X		4.6 Searching
X		X			4.7 Sort/Merge

Appendix B. Capability Definitions

Taxonomy Outline

We provide the following abbreviated taxonomy to aid understanding of and provide a quick reference for the lengthy capability definitions given in the next section.

```
1 Management Capabilities
1.1 Computer System Management
1.2 Configuration Management
1.3 Process Management
1.4 Project Management
1.5 Project Repository Management

2 Development Engineering Capabilities
2.1 Concept Development
2.2 Environment Growing
2.3 System Development
2.4 Productization
2.5 Operational Maintenance

3 Communication Capabilities
3.1 Help Facility
3.2 Project Communication
3.3 User Interface

4 Global Object Manipulation Capabilities
4.1 Browsing
4.2 Comparison
4.3 Editing
4.4 Formatting
4.5 Output
4.6 Searching
4.7 Sort/Merge
4.8 Spelling Checking
```

Definitions

In the spirit of reuse and of consistent use of terminology, many of these definitions have been given previously in (Goos83), (Kean85), (Lyons86), (STONE80), (TCSEC83), and (Weide87).

Note: To help distinguish which of the following belong to our definition of the SFLC, each phase of the SFLC is labelled with either *phase* or *subphase*. We distinguish capabilities by the wording *capability* or *set of capabilities*.

1 Management Capabilities

This section of the taxonomy discusses those capabilities that permit establishment, administration, tracking, and control of system development projects. Examples of management capabilities are computer system management, configuration management, process management, project management, and project repository management.

1.1 Computer System Management

Computer System Management is the set of capabilities that permit establishment, administration, tracking, and control of computer systems.

```
1.1.1 Access Management
1.1.1.1 Access Control
1.1.1.1.1 Discretionary Control
1.1.1.1.2 Group Definition
1.1.1.1.3 Mandatory Control
1.1.1.1.4 Object Control
1.1.1.1.5 Process Control
```

Access Management -- the set of capabilities that govern access to a computer system and its objects and processes.

Access Control -- the set of capabilities that govern access to objects and processes.

Discretionary Control -- the capability to restrict access to objects based on the identity of users and/or groups to which they belong. The controls are discretionary in the sense that a subject with certain access permission is capable of passing that permission (perhaps indirectly) to any other subject.

Group Definition -- the capability to define groups of persons who have the same access privileges.

Mandatory Control -- the capability to restrict access based directly on a comparison of a user's clearance or authorization for the information and the classification or sensitivity designation of the information being sought.

Object Control -- the capability to control access to objects.

Process Control -- the capability to control access to processes.

```
1.1.2 System Resource Management
1.1.2.1 Accounting Information Collection
1.1.2.2 Archival, Backup, and Recovery
1.1.2.3 Configuration Modification
1.1.2.4 Workload Monitoring
```

System Resource Management -- the set of capabilities that permit control of system resources.

Accounting Information Collection -- the capability to collect information that can be used in billing for used resources.

Archival, Backup and Recovery -- the capability to archive the system, back the system up, and recover to a previous version of the system.

Configuration Modification -- the capability to change the computer system configuration. For example, more virtual memory or a new device might be added.

Workload Monitoring -- the capability to view how system resources are used.

```
1.1.3 User Account Management
1.1.3.1 Initial Working Directory Creation
1.1.3.2 Login/Logout Macro Creation
1.1.3.3 User Account Group Maintenance
1.1.3.4 User Account Maintenance
```

User Account Management -- the set of capabilities that aid in the management of user accounts.

Initial Working Directory Creation -- the capability to automatically create an initial workspace for a newly defined user.

Login/Logout Macro Creation -- the capability to automatically create programs that run upon user logon and logoff.

User Account Group Maintenance -- the set of capabilities that allow user groups to be established, deleted, and otherwise modified.

User Account Maintenance -- the set of capabilities that allow user accounts to be established, deleted, and otherwise modified.

1.2 Configuration Management

Configuration Management is the set of capabilities that define, track, and control various versions of a system. These capabilities include configuration identification, configuration control, and configuration status accounting.

```
1.2.1 Configuration Control
1.2.1.1 Baseline Definition
1.2.1.2 Foreign Object Import
1.2.1.3 Multiple Development Path Creation
1.2.1.4 Source Code Support
1.2.1.4.1 Alternate Implementation Specification
1.2.1.4.2 Meta-Package/Subsystem Support
```

Configuration Control -- the set of capabilities that permit definition, identification, assembling, tracking, and distributing configurations. These capabilities also provide for the definition of the policies, rules, and procedures for initial acceptance of and making changes to controlled items.

Baseline Definition -- the capability to identify a collection of objects that will be formally placed under configuration control and that will be used to track future progress.

Foreign Object Import -- the capability to bring externally defined objects (e.g., subcontractor products) into a repository.

Multiple Development Path Creation -- the capability to create parallel development paths from a single baseline.

Source Code Support -- the set of capabilities that provide support for specifying and controlling various source code configurations.

Alternate Implementation Specification -- the capability to specify more than one implementation (e.g., in Ada, bodies) for a given specification.

Meta-Package/Subsystem Support -- the capability to organize source code units into collections for ease of handling and control. For example, in Ada, a collection of packages (a meta-package or subsystem) is often desired.

```
1.2.1.5 System Build Definition
1.2.1.5.1 Object Dependency Specification
1.2.1.5.2 Object Specification
1.2.1.5.3 Translator Specification
```

System Build Definition -- the set of capabilities that define the objects and processes necessary to create a configuration.

Object Dependency Specification -- the capability to specify the relationships among the objects to be built (and consequently the order in which the objects must be processed.)

Object Specification -- the capability to specify those objects to be built into a configuration.

Translator Specification -- the capability to specify the processors, rules, and options that will be used to process the objects in the configuration (e.g., spelling checker and its dictionary, linker and its options).

```
1.2.1.6 System Build
1.2.1.6.1 Current Default Build
1.2.1.6.2 History Capture
1.2.1.6.3 Hybrid Build
1.2.1.6.4 Inconsistent System Warning
1.2.1.6.5 Partial/Incomplete Build
1.2.1.6.6 Specific Change or Problem Build
1.2.1.6.7 Specific Release Build (Rebuild)
```

System Build -- the set of capabilities that support assembling a system.

Current Default Build -- the capability to retain the definition of a current build and to use this definition when a specific build is not requested.

History Capture -- the capability to record how objects are produced and all information relevant to their production.

Hybrid Build -- the capability to build a system based on a system build definition which mixes objects/modules from different releases.

Inconsistent System Warning -- the capability to warn the system builder when an inconsistency has been discovered during the build process.

Partial/Incomplete Build -- the capability to build systems where all of the pieces are not complete.

Specific Change or Problem Build -- the capability to rebuild the current system to include an enhancement provided by a change or problem fix.

Specific Release Build (Rebuild) -- the capability to build/rebuild a specific system release.

```
1.2.2 Product Release Control
1.2.2.1 Release Generation
1.2.2.2 Release History Capture
1.2.2.2.1 Differences Among Releases
1.2.2.2.2 Errors Reported/Fixed by Release
1.2.2.2.3 Hardware Configuration for Release
1.2.2.2.4 Locations of Release
1.2.2.2.5 Release Identification
1.2.2.2.6 Software Configuration for Release
```

Product Release Control -- the set of capabilities that support the generation and release of a system.

Release Generation -- the capability to define a uniquely identified release.

Release History Capture -- the set of capabilities that record how a release is produced and all the information pertaining to its production.

Differences Among Releases -- the capability to identify the changes in a release as compared to another release.

Errors Reported/Fixed by Release -- the capability to identify the problems reported against and fixed by a release.

Hardware Configuration for Release -- the capability to identify and record the hardware configuration necessary to support a system release.

Locations of Release -- the capability to record where releases of a system were sent.

Release Identification -- the capability to uniquely identify each release.

Software Configuration for Release -- the capability to identify and record the software configuration necessary to support a system release.

```
1.2.3 Version Control
1.2.3.1 Comparison
1.2.3.2 Creation
1.2.3.2.1 Initial
1.2.3.2.2 Successor
1.2.3.2.3 Variant
1.2.3.3 History Capture
1.2.3.4 Merging
1.2.3.5 Reservation
1.2.3.6 Retrieval
1.2.3.6.1 Explicit
1.2.3.6.2 Dynamic
1.2.3.6.3 Referential
```

Version Control -- the set of capabilities that allow versions of objects to be maintained, tracked, and controlled.

Comparison -- the capability to compare two versions and indicate where they differ.

Creation -- the set of capabilities that allow new versions of objects to be created.

Initial -- the capability that identifies an object as newly created and places it under control.

Successor -- the capability to create a version of an object designed to replace the original object (usually includes one or more enhancements or bug fixes).

Variant (e.g., optimized module) -- the capability to create an alternative version of an object (e.g., one destined for a different target or an optimized version).

History Capture -- the capability to record how an object has evolved and all the information pertaining to its evolution.

Merging -- the capability to merge two or more variants of an object so that the result contains the most current updates of all the variants.

Reservation -- the capability that controls object modification by forcing the modifier to reserve the object from the controlled repository before it can be changed. When an object is reserved, no one other than the reserver can modify it.

Retrieval -- the capability to retrieve (recreate) any version from the project repository.

Explicit -- the capability to retrieve a version by its unique identifier.

Dynamic -- the capability to retrieve those versions specified by a number of dynamic attributes (e.g., most current version, previous version).

Referential -- the capability to retrieve a version by referencing a configuration which includes that version (e.g., the version that was shipped in Release 3.1).

```
1.2.4 Problem and Change Management
1.2.4.1 Assignment and Authorization
1.2.4.2 Association with Test Cases
1.2.4.3 Cataloging
1.2.4.4 Report/Status Generation
1.2.4.5 Resolution Text Capture
```

Problem and Change Management -- the set of capabilities that control, track, and maintain Problem Reports (PR) and Change Requests (CR).

Assignment and Authorization -- the capability to assign/reassign a PR/CR to a person or group.

Association with Test Cases -- the capability of mapping a PR/CR to the test case(s) from which it was generated.

Cataloging -- the capability of sorting and classifying PRs/CRs by some defined method.

Report/Status Generation -- the capability of generating various reports pertaining to the status of PRs/CRs.

Resolution Text Capture -- the capability to record the resolution text for a particular PR/CR.

1.3 Software Process Management

Software Process Management is the set of capabilities that allow the software development process used in a particular software engineering environment to be specified and controlled.

```
1.3.1 Software Process Definition
1.3.1.1 Software Process Type Definition
1.3.1.2 Software Process Instance Creation
1.3.1.3 Invocation Condition Specification
1.3.2 Process Sequencing
1.3.3 Consistency Checking and Updating
```

Software Process Definition -- the set of capabilities to define a process in the life cycle.

Software Process Type Definition -- the capability to define the characteristics of type of processes.

Software Process Instance Creation -- the capability to define a single instance of a process type.

Invocation Condition Specification -- the capability to specify for a process or process type the conditions upon which it may be invoked.

Process Sequencing -- the capability to specify (when not implied by the invocation conditions) the sequence of processes in the life cycle.

Consistency Checking and Updating -- the capability to check whether invoking a process would cause objects to become inconsistent, to provide a warning in this case, and to update any consistency information changed as a result of invoking a process.

1.4 Project Management

Project Management is the control and tracking of a project. Its capabilities generally include allocating resources to the project, specifying group and individual tasks, tracking progress, scheduling, and evaluating the quality of the project.

```
1.4.1 Financial Management
1.4.1.1 Cost Actuals Collection
1.4.1.2 Cost Assessment
1.4.1.3 Cost Estimation
1.4.1.4 Work Breakdown Structure
```

Financial Management -- the set of capabilities that support the financial aspects of managing a system/software development project.

Cost Actuals Collection -- the capability to enter and retain actual cost accounting data.

Cost Assessment -- the capability to determine accrued resource costs.

Cost Estimation -- the capability to estimate project cost based on historical data, simulations, and other data. Elements of cost are typically labor, computer time, materials, travel, etc.

Work Breakdown Structure -- the capability to break the project into hierarchic refinements of detail that define work to be done into short, manageable tasks with quantifiable inputs, outputs, schedules and assigned responsibilities.

```
1.4.2 Project Definition
1.4.2.1 Organization Definition
1.4.2.1.1 Responsibility Assignment
1.4.2.2 Project Initiation
1.4.2.3 Project Deletion
```

Project Definition -- the set of capabilities necessary to identify and maintain projects.

Organization Definition -- the capability to define the members of a project team and their relationships (for generation of account and access groups, distribution lists, authorization procedures, etc.).

Responsibility Assignment -- the capability to assign responsibility for completion of tasks to individuals or groups.

Project Initiation -- the capability to make a new project known to a computer system, including creation of initial project information and allocation of initial system resources.

Project Deletion -- the capability to delete a project and its corresponding project information and to revoke its resources.

```
1.4.3 Quality Management
1.4.3.1 Auditing
1.4.3.2 Evaluating
1.4.3.3 Project Procedures Definition
1.4.3.4 Project Standards Definition
1.4.3.5 Quality Factor Definition
1.4.3.6 Quality Prediction
```

Quality Management -- the capabilities that ensure that the project produces quality products.

Auditing -- the capability to ensure that predefined rules were followed in the creation/development of an object.

Evaluating -- the capabilities for evaluating a product against prescribed standards.

Project Procedures Definition -- the capability to identify standard procedures to be used by the project (e.g., change authorization procedure, requests for computer resources).

Project Standards Definition -- the capability to identify standards or standards enforcement processors to be used by the project (e.g., naming conventions).

Quality Factor Definition -- the capability to define and prioritize the quality factors required for a given application. Potential quality factors include: efficiency, integrity, reliability, survivability, usability, correctness, maintainability, verifiability, expandability, flexibility, interoperability, portability, and reusability.

Quality Prediction -- the capability to estimate based on historical data and current values for quality factors the quality of a given object.

1.4.4 Resource Management
1.4.4.1 Assessment
1.4.4.2 Balancing
1.4.4.3 Definition
1.4.4.4 Estimation

Resource Management -- the set of capabilities that provide for resource allocation and tracking for a system/software project. A resource is any thing, person, action etc. necessary to complete a given task. Examples of resources include manpower, computer memory, and CPU.

Assessment -- the capability to determine the amount of resources currently in use.

Balancing -- the capability to make planned and actual resource use conform, usual encompassing acquisition of more resources, swapping resources among tasks, rescheduling resource use, and trading abundant resources for a scarce resources.

Definition -- the capability to define new resources to be tracked.

Estimation -- the capability to estimate project resource use based on historical data, simulations, and other data.

1.4.5 Risk Management
1.4.5.1 Object to Risk Category Association
1.4.5.2 Risk Category Definition
1.4.5.3 Risk Management Text Definition
1.4.5.3.1 Category
1.4.5.3.2 Object
1.4.5.4 Status Reporting

Risk Management -- the set of capabilities that provide for defining, tracking, and planning for project risk.

Object to Risk Category Association -- the capability to define relationships between objects and any associated risk categories.

Risk Category Definition -- the capability to define a category of risk (e.g., key personnel, inclement weather).

Risk Management Text Definition -- the capabilities to define and record plans for handling risks.

Category -- the capabilities to define and record plans for handling risk categories.

Object -- the capabilities to define and record plans for handling the risks associated with producing certain objects or with performing certain tasks.

Status Reporting -- the capability to report the plans for handling risks and the current assessment of those risks.

```
1.4.6 Schedule Management
1.4.6.1 Baseline Maintenance
1.4.6.2 Calendar Management
1.4.6.2.1 Individual
1.4.6.2.2 Project
1.4.6.3 Contingency Analysis
1.4.6.4 Critical Path Definition
1.4.6.5 Earned Value Calculation
1.4.6.6 Milestone Definition
1.4.6.7 Schedule Assessment
1.4.6.8 Schedule Updating and Recalculation
```

Schedule Management -- the set of capabilities that allow schedules to be defined, maintained, and assessed.

Baseline Maintenance -- the capability to record and update a baseline schedule.

Calendar Management -- the capability to maintain a calendar.

Individual -- the capability to maintain a calendar for individual tasks or persons.

Project -- the capability to maintain a calendar for the project. Project calendars typically include deliverables, deadlines, milestones, key reviews, and may include parts of individual task calendars.

Contingency Analysis -- the capability to modify parts of a schedule to analyze the impact of the change(s). This also includes the capabilities to analyze slack time, resource balancing, rescheduling, and other facets of contingency planning.

Critical Path Definition -- the capability to identify and track those schedule elements determined to be most critical to completing on the scheduled date.

Earned Value Calculation -- the capability to provide an assessment of earned value to date for a given project.

Milestone Definition -- the capability to define and track progress toward milestones.

Schedule Assessment -- the capability to review the current schedule against the baseline to show project status.

Schedule Updating and Recalculation -- the capability to update a schedule (e.g., actual completion dates, additional tasks, resources consumed) and recalculate the schedule.

```
1.4.7 Tracking
1.4.7.1 Cost
1.4.7.2 Project Data
1.4.7.3 Quality
1.4.7.4 Resource Use
1.4.7.5 Technical Performance
```

Tracking -- the capability to record and analyze data on various facets of a project.

Cost -- the capability to record and analyze cost data.

Project Data -- the capability to track the location of all project data.

Quality -- the capability to record and analyze quality data.

Resource Use -- the capability to record and analyze resource use data.

Technical Performance -- the capability to record and analyze aspects of technical performance (e.g., lines of code produced, errors discovered).

1.5 Project Repository Management

Project Repository Management is that set of capabilities that provide for the creation, modification, and administration of every object created during the life of a project.

```
1.5.1 Repository Administration
1.5.1.1 Concurrent Access
1.5.1.2 Maintenance
1.5.1.3 Navigation Mechanism Definition
1.5.1.3.1 Hierarchical
1.5.1.3.2 Networked
1.5.1.4 Query Language
1.5.1.5 Reuse Repository Management
1.5.1.5.1 Acceptance Criteria Establishment
1.5.1.5.2 Cataloging
1.5.1.5.3 Search/Retrieval
```

Repository Administration -- the set of basic capabilities necessary to the functioning of a repository.

Concurrent Access -- the capability of more than one user to access the repository at the same time.

Maintenance -- the capability of maintaining the repository, its access methods, and any objects it contains.

Navigation Mechanism Definition -- the set capabilities that define how users may move from object to object in the repository.

Hierarchical -- the capability to move in a hierarchical fashion; each object is related to others in a tree like fashion.

Networked -- the capability to move in a network fashion; each object is related to others in a graph like fashion.

Query Language -- the capability to search for objects based upon certain attributes or criteria.

Reuse Repository Management -- the set of capabilities necessary to establish and maintain a repository of reusable components.

Acceptance Criteria Establishment -- the capability to set minimal criteria for the acceptance of an object into the repository.

Cataloging -- the capability to maintain a directory of those components contained in a reuse repository.

Search/Retrieval -- the ability to locate and extract a component from a reuse repository.

1.5.2 Documentation Management
1.5.2.1 Document Template Definition
1.5.2.2 Document Type Definition
1.5.2.3 Integrated Text and Graphics
1.5.2.3.1 User Interface Display Capture
1.5.2.4 Processing
1.5.2.5 Rerelease Analysis
1.5.2.5.1 Change Bars
1.5.2.5.2 Page Renumbering/Insertion
1.5.2.6 Source Code Publication
1.5.2.6.1 Formatting
1.5.2.6.2 Indexing/Headers/Table of Contents
1.5.2.6.3 Macro Processing
1.5.2.6.4 Pragma Sensitivity

Documentation Management -- the set of capabilities that govern the definition, storage, formatting, and tracking of documentation.

Document Template Definition -- the capability to define document templates to serve as bases and as guidelines for document preparation.

Document Type Definition -- the capability to define various document types, their relationships to other objects, etc.

Integrated Text and Graphics -- the capability support both text and graphics in a single document.

User Interface Display Capture -- the capability to capture a picture of the user interface display and use it in documentation.

Processing -- the capability to format a document for printing on an output device.

Rerelease Analysis -- the capability to analyze an updated document against its previous version (or versions).

Change Bars -- the capability to flag changed lines with bars that appear in the margin.

Page Renumbering/Insertion -- the capability to produce renumbered pages for insertion (as errata or addenda) into existing documents and the capability to produce a list of modified and deleted pages.

Source Code Publication -- the set of capabilities that provide for the publication of source code listings.

Formatting -- the capability to format source code according to user or predefined conventions.

Indexing/Headers/Table of Contents -- the capability to include indices, headers, and tables of contents in source code listings.

Macro Processing -- the capability to define and invoke macros.

Pragma Sensitivity -- the capability of the source code publication system to respond to the Ada pragmas List and Page.

1.5.3 Object Management
1.5.3.1 Consistency Management
1.5.3.1.1 Analysis
1.5.3.1.2 Definition
1.5.3.1.3 Change Impact Analysis
1.5.3.1.4 Change Information Collection
1.5.3.1.5 Reporting

Object Management -- the set of capabilities that allow a software engineering environment to manage, track, and maintain objects. Examples of objects are documents, source code, memos, change requests, etc.

Consistency Management -- the set of capabilities that allow a software engineering environment to maintain or determine the consistency of any of its objects.

Analysis -- the capability to analyze the objects stored in the repository and their relationships to determine the consistency of the system.

Definition -- the capability that allows consistency to be defined for each type of object defined in the software engineering environment.

Change Impact Analysis -- the capability to analyze the objects stored in the repository and their relationships to determine the impact of changing an object.

Change Information Collection -- the capability to collect and maintain historical data about changes and revisions to objects for input to consistency analysis.

Reporting -- the capability to report the consistency of objects in the system, along with the definition of consistency for each type of object.

```
1.5.3.2 Maintenance
1.5.3.2.1 Creation
1.5.3.2.1.1 Attribute Definition
1.5.3.2.1.2 Relationship Definition
1.5.3.2.1.2.1 Allowed Processes Definition
1.5.3.2.1.2.1 Standards Processing Definition
1.5.3.2.2 Deletion
1.5.3.2.3 Modification
```

Maintenance -- the set of capabilities that allows objects to created, deleted, and modified.

Creation -- the capability that allows new objects and types of objects to be defined to the system.

Attribute Definition -- the capability to specify attributes of an object (e.g., object type = Ada source) and of relationships.

Relationship Definition -- the capability to specify relationships between objects or types of objects (e.g., dependent upon, parent of).

Allowed Processes Definition. -- the capability to specify, for an object or type of object, those processes which are allowed to operate upon it.

Standards Processing Definition -- the capability to specify, for any object or type of object, those processes which will be automatically invoked to check the conformity of the object to project standards.

Deletion -- the capability to delete an object or an object type definition.

Modification -- the capability to modify the definition of an object or object type.

```
1.5.3.3 Object Dictionary Management
1.5.3.3.1 Cross Reference
1.5.3.3.2 Interactive Query
1.5.3.3.3 Navigation
1.5.3.3.4 Object and Relationship Display
1.5.3.3.5 Reporting
1.5.3.4 Owner Assignment
```

Object Dictionary Management -- the set of capabilities that give users visibility to objects, their use, definitions, relations etc.

Cross Reference -- the capability to determine from the project repository all the uses (including the definition) of an object or object type.

Interactive Query -- the capability to interactively search the repository for objects that match certain criteria.

Navigation -- the capability to move about the various objects in the project repository based upon the relationship network (e.g., go to the testing history of a given unit).

Object and Relationship Display -- the capability to (graphically) display objects and their relationship networks.

Reporting -- the capability to provide reports based on queries of the repository.

Owner Assignment -- the capability to assign an owner to every object.

```
1.5.3.5 Program Library (Ada) Management
1.5.3.5.1 Creation/Deletion/Maintenance
1.5.3.5.2 Foreign Code Import
1.5.3.5.3 Navigation
1.5.3.5.4 Program Library Relationship Specification
1.5.3.5.5 Query
1.5.3.5.5.1 Completeness/Recompilation Status
1.5.3.5.5.2 Relationships
1.5.3.5.5.3 Unit Information
1.5.3.6 Retrieval
```

Program Library (Ada) Management -- the set of capabilities that support Ada program libraries

Creation/Deletion/Maintenance -- the capability to create, delete, or otherwise modify a given program library.

Foreign Code Import -- the capability to bring in Ada code from outside sources (e.g., from subcontractors).

Navigation -- the capability to move about the program library based upon the relationship network of the units (e.g., go to a particular "with"ed unit).

Program Library Relationship Specification -- for those systems that do not have monolithic program libraries, the capability to specify the relationships among the various libraries (e.g., if the same unit name appears in two libraries, which will take precedence?).

Query -- the capability to interactively search the library for units (and perhaps smaller units like types and variables) that match certain criteria.

Completeness/Recompilation Status -- the capability to report which units are complete or incomplete (e.g., separate subunits that are not yet coded) and which units need to be recompiled.

Relationships -- the capability to report the relationship network for a given unit or set of units.

Unit Information -- the capability to report general component information (e.g., name of unit, time stamp of last compile, time stamp of last modification, etc.).

Retrieval -- the capability to retrieve the Ada version (i.e., not all systems store units in source format) of a unit.

2 Development Engineering Capabilities

This section of the taxonomy describes the capabilities that support the Software First Life Cycle, the SFLC.

2.1 Concept Development (*)

Concept Development is the first phase of the SFLC. In this phase, the customer and the contractor(s) arrive at an initial understanding of the desired system concept.

2.2 Environment Growing (*)

Environment Growing is the second phase of the SFLC. In this phase, the contractor assembles a Software Engineering Environment that meets the domain specific requirements of the development.

```
2.2.1 Environment Tailoring (*)
2.2.2 Repository Assembly (*)
```

Environment Tailoring -- the first of two parallel subphases of Environment Growing in which a particular SEE is instantiated and customized. Typical of those items to be specified are the project organization, the software development process, the allowable states and processes for a given development, and document types and formats.

Repository Assembly -- the second of two parallel subphases of Environment Growing in which the application specific repository is assembled from various sources. Alternatively, the application specific repository may just be a view of a larger repository and may not involve assembly at all.

2.3 System Development (*)

System Development is the third phase of the SFLC. In this phase, the system is developed from the operational concept to a working prototype(s) which meets the customer's need.

```
2.3.1 System Architecture Definition (*)
2.3.1.1 Software Architecture Definition (*)
2.3.1.1.1 Interface Selection/Definition (*)
2.3.1.1.1.1 Architecture Rationale Capture
2.3.1.1.1.2 Control Flow Definition
2.3.1.1.1.3 Data Flow Definition
2.3.1.1.1.4 Object and Operation Definition
2.3.1.1.1.5 Program Unit Interface Definition
2.3.1.1.1.5.1 Graphical System Description
2.3.1.1.1.5.2 System Description to Ada Conversion
2.3.1.1.2 Interface/Architecture Evaluation (*)
```

System Architecture Definition -- the first of three parallel subphases of System Development. During this phase, the software and hardware architectures are developed and refined.

Software Architecture Definition -- the first of two overlapping subphases of System Architecture Definition. The capabilities of this phase define the initial software architecture definition, refine it, and produce a final definition.

Interface Selection/Definition -- the initial subphase of the Software Architecture Definition, this phase encompasses selecting (or defining when such do not exist) those interfaces that are key to the prototype. As the prototypes provided feedback on the interfaces, they are adjusted accordingly.

Architecture Rationale Capture -- the capability to record any decisions made during the definition of the software architecture and the reasons for these decisions.

Control Flow Definition -- the capability to define the sequence of execution of the various parts of the software.

Data Flow Definition -- the capability to define the sequence of references to data during the execution of the software.

Object and Operation Definition -- the capability to define those basic objects to be provided by the software system, their characteristics, attributes, normal and abnormal behavior, and the operations that are allowed upon these objects.

Program Unit Interface Definition -- the capability to formally specify the external interfaces of a program unit; from this point onward others may depend upon these interfaces.

Graphical System Description -- the capability to describe a system and its various flows graphically instead of textually.

System Description to Ada Conversion -- the capability to take a system description, whether it be graphical or textual, and convert it to Ada code. This does not mean that the code will be complete and executable, only that the more tedious aspects of conversion are done automatically.

Interface/Architecture Evaluation -- the second subphase of the Software Architecture Definition, this phase encompasses evaluating the success of a particular set of interfaces (defining an architecture) and recording lessons learned as a result of implementing these interfaces in a prototype.

```
2.3.1.2 Hardware Architecture Definition (*)
2.3.1.2.1 Resource/Trade-off Studies (*)
2.3.1.2.1.1 Simulation and Modelling
2.3.1.2.2 Hardware Definitization (*)
2.3.1.2.3 Hardware Selection (*)
```

Hardware Architecture Definition -- the second of two overlapping subphases of System Architecture Definition. The capabilities of this phase define the hardware to support the system software.

Resource/Trade-off Studies -- the first subphase of Hardware Architecture Definition. This phase provides the capabilities to study (e.g., simulate) the hardware resources required by the delivered system.

Simulation and Modelling -- the capability to represent selected characteristics of one physical or abstract system by another system.

Hardware Definitization -- the second subphase of Hardware Architecture Definition. This phase provides the capabilities to refine the hardware resources required by the evolving system prototypes.

Hardware Selection -- the third subphase of Hardware Architecture Definition and the final act of System Architecture Definition. This phase provides the capability to choose hardware to support the software architecture once the software architecture is defined.

```
2.3.2 System Capability Definition (*)
2.3.2.1 Preliminary Capability Statement (*)
2.3.2.2 Capability Definitization (*)
2.3.2.3 Formal Capability Statement (*)
```

System Capability Definition -- the second of the three parallel subphases of System Development, this phase aims to translate customer needs into capabilities that the developers can implement.

Preliminary Capability Statement -- the initial subphase of System Capability Definition. During this subphase, a statement of capabilities is derived from the user's concept of the system.

Capability Definitization -- the middle subphase of System Capability Definition. During this subphase, the capabilities desired are defined and clarified.

Formal Capability Statement -- the final subphase of System Capability Definition. During this subphase, the final statement of the capabilities to be included in the delivered system is defined.

```
2.3.3 Prototype Competition (*)
2.3.3.1 Alternative Formulation (*)
2.3.3.1.1 Alternative Consideration (*)
2.3.3.1.2 Breakthrough Consideration (*)
2.3.3.1.3 Alternative Evaluation (*)
2.3.3.1.3.1 Evaluation Results Capture
2.3.3.1.3.2 Selection Rationale Capture
```

Prototype Competition -- the third of the three parallel subphases of System Development, this phase sees prototypes competing against one another in the search for the best possible system.

Alternative Formulation -- the first of three subphases of Prototype Competition, this subphase generates alternative solutions to be implemented as prototypes.

Alternative Consideration -- the first of three subphases of Alternative Formulation, this phase encompasses th. process of determining and elaborating upon candidate solutions.

Breakthrough Consideration -- the second of three subphases of Alternative Formulation, this phase encompasses the process of searching for novel solutions.

Alternative Evaluation -- the third of three subphases of Alternative Formulation, this phase encompasses the process of evaluating the candidate solutions and selecting one or more to be prototyped.

Evaluation Results Capture -- the capability to record the results of evaluating each prototype, primarily in the form of lessons learned.

Selection Rationale Capture -- the capability to record the reasons for selecting a particular prototype over other candidates.

```
2.3.3.2 Prototyping (*)
2.3.3.2.1 Component Acquisition (*)
2.3.3.2.1.1 Component Reuse Analysis (*)
2.3.3.2.1.1.1 Component Selection Guidance
2.3.3.2.1.2 Component Development/Refinement (*)
2.3.3.2.1.2.1 Component Prototyping (*)
                (see 2.3.3.2 Prototyping (recursive))
```

Prototyping -- the second of three subphases of Prototype Competition, this subphase provides the capabilities to support the iterative process of exploratory programming.

Component Acquisition -- the first of four sequential subphases of the Prototyping subphase, this subphase includes the capabilities for collecting the necessary components to build a given prototype.

Component Reuse Analysis -- the first subphase of Component Acquisition, in which the repository is searched for candidate components and these components are examined for their applicability to a given prototype.

Component Selection Guidance -- the capability to provide (expert) assistance in choosing a component to meet certain needs.

Component Development/Refinement -- the second subphase of Component Acquisition, entered only if components are not found in the repository or if they are found to be inadequate, in which reusable components are built or rebuilt for insertion into the reuse repository and for use in a given prototype.

Component Prototyping -- the first subphase of Component Development/Refinement. During this subphase, exploratory programming is aimed at developing reusable components.

```
2.3.3.2.1.2.2 Component Engineering (*)
2.3.3.2.1.2.2.1 Specification Development (*)
2.3.3.2.1.2.2.1.1 Component Development Folder
2.3.3.2.1.2.2.1.2 Object and Operation Definition/Redefinition
2.3.3.2.1.2.2.1.3 Program Unit Interface Definition/Redefinition
2.3.3.2.1.2.2.1.4 Static Analysis
2.3.3.2.1.2.2.1.4.1 Type Analysis
2.3.3.2.1.2.2.1.4.2 Unit Analysis
2.3.3.2.1.2.2.1.5 Translation
                    (see 2.3.3.2.1.2.2.2.2 Translation)
```

Component Engineering -- the second and final subphase of Component Development/Refinement. During this subphase, reusable components are built, either from scratch or from some existing model and often based on the results of component prototypes.

Specification Development -- the first subphase of Component Engineering and the one that provides capabilities for creating the external interfaces of a component.

Component Development Folder -- the capability to record events, notes, and other items of interest during the development of a component. For example, a component development folder might indicate how many changes have been made to a component, who made the changes, the rationale for the changes, the size of the module at given times, etc.

Object and Operation Definition/Redefinition -- the capability to define the basic object(s) that the component will provide and the permitted operations on the object(s).

Program Unit Interface Definition/Redefinition -- the capability to formally record the external interface of the component.

Static Analysis -- the capability to provide operations against an object that analyze the object without regard to the dynamic nature of the object (e.g., whether or not it will execute).

Type Analysis -- the capability to detect similarities among types for potential merging.

Unit Analysis -- the capability to detect whether the units attributed to an object (e.g., meters/second) are correct and consistent.

```
2.3.3.2.1.2.2.2 Implementation Development (*)
2.3.3.2.1.2.2.2.1 Body Development Support
2.3.3.2.1.2.2.2.1.1 Body Stub Generation
2.3.3.2.1.2.2.2.1.2 Body Template Generation
2.3.3.2.1.2.2.2.1.3 Commentary Transfer
2.3.3.2.1.2.2.2.1.4 Inline-Separate Unit Conversion
```

Implementation Development -- the second subphase of Component Engineering and the one that provides capabilities for creating the implementation dependent portions of a component. In Ada, this would be the creation of the bodies.

Body Development Support -- the set of capabilities necessary to support the development of the bodies from their corresponding specifications.

Body Stub Generation -- the capability to create a (nominally) executable body from a specification.

Body Template Generation -- the capability to create a body for a given specification which has templates for each of the subprograms/entries, but which is not intended to be executable.

Commentary Transfer -- the capability to transfer selected commentary (e.g., behavioral descriptions) from the specification to the body.

Inline-Separate Unit Conversion -- the capability to make an inline Ada unit separate and vice-versa.

```
2.3.3.2.1.2.2.2.2 Translation
2.3.3.2.1.2.2.2.2.1 Assembling
2.3.3.2.1.2.2.2.2.1.1 Macro Expansion
2.3.3.2.1.2.2.2.2.1.2 Structure Preprocessing
2.3.3.2.1.2.2.2.2.2 Compilation
2.3.3.2.1.2.2.2.2.2.1 Cross
2.3.3.2.1.2.2.2.2.2.2 Incremental
2.3.3.2.1.2.2.2.2.2.3 Intermediate Language Generation
2.3.3.2.1.2.2.2.2.2.4 Recompilation of Withed and Dependent Units
2.3.3.2.1.2.2.2.2.2.5 Semantic Checking
2.3.3.2.1.2.2.2.2.2.6 Syntax Checking
2.3.3.2.1.2.2.2.2.3 Conversion
2.3.3.2.1.2.2.2.2.4 Decompilation
2.3.3.2.1.2.2.2.2.5 Disassembly
2.3.3.2.1.2.2.2.2.6 Source Reconstruction
```

Translation -- the set of capabilities to convert from one computer language to another.

Assembling -- the capability to convert machine-level (assembly) code to object-level code, ready for linking and execution.

Macro Expansion -- the capability to preprocess assembly code to replace macro calls with equivalent assembly code.

Structure Preprocessing -- the capability to replace higher level structures in assembly code with equivalent assembly code.

Compilation -- the capability to convert higher order code (e.g., Ada) into lower-order code (e.g., assembly, object, or intermediate code).

Cross-Compilation -- the capability for a compiler running on one machine to produce code targeted to execute on another machine, often of a different type.

Incremental Compilation -- the capability to compile only those parts of a program affected by a change.

Intermediate Language Generation -- the capability to generate an intermediate form of a language (e.g., DIANA).

Recompilation of Withed and Dependent Units -- the capability to automatically recompile those Ada program units that are affected by a given change.

Semantic Checking -- the capability to check the semantics of a program without performing any translation.

Syntax Checking -- the capability to check the syntax of a program without performing any translation.

Conversion -- the capability to convert between any two languages, typically languages of the same level (e.g., Ada and C).

Decompilation -- the capability to convert machine code into a high-order language (i.e., back into the same language from which the machine code was created).

Disassembly -- the capability to convert machine code into assembly language.

Source Reconstruction -- the capability to convert from intermediate language back to a high-level language.

```
2.3.3.2.1.2.2.2.3 Debugging
2.3.3.2.1.2.2.2.3.1 Cross-Debugging
2.3.3.2.1.2.2.2.3.1.1 Target Machine Instruction Simulation
2.3.3.2.1.2.2.2.3.2 Machine Level Debugging
2.3.3.2.1.2.2.2.3.3 Simultaneous Symbolic and Machine Level Debugging
```

Debugging -- the capability to discover, analyze, and correct suspected errors in a program.

Cross-Debugging -- the capability to use a debugger executing on one machine to debug a program executing on another machine.

Target Machine Instruction Simulation -- the capability for a debugger on one machine to simulate the execution of a program on another machine.

Machine Level Debugging -- the capability to debug at a machine language level.

Simultaneous Symbolic and Machine Level Debugging -- the capability to debug using both symbolic and machine level code and commands at the same time.

```
2.3.3.2.1.2.2.2.3.4 Symbolic Debugging
2.3.3.2.1.2.2.2.3.4.1 Breakpoint Setting/Clearing
2.3.3.2.1.2.2.2.3.4.2 Execution Control
2.3.3.2.1.2.2.2.3.4.3 Program State Modification
2.3.3.2.1.2.2.2.3.4.4 Program State Query
2.3.3.2.1.2.2.2.3.5 Memory Dump
```

Symbolic Debugging -- the capability to debug at a high-order language level.

Breakpoint Setting/Clearing -- the capability to cause a program to halt execution at given locations, for example, on exceptions, statements, and rendezvous.

Execution Control -- the capability to control the execution of statements of a program while in the debugger.

Program State Modification -- the capability to change the state of an executing program.

Program State Query -- the capability to view information about the state of an executing program.

Memory Dump -- the capability to capture the contents of memory (or a part of memory) for use in debugging.

```
2.3.3.2.1.2.2.2.4 Linking/Loading
2.3.3.2.1.2.2.2.4.1 Cross-Linking
2.3.3.2.1.2.2.2.4.2 Partial Linking
```

Linking/Loading -- the capability to produce a load or executable module from one or more independently produced load or executable modules.

Cross-Linking -- the capability for one machine to produce a load or executable module for another machine

Partial Linking -- the capability to produce load or executable modules from parts of systems. For example, producing a load module from a database interface and placing this module in a library is preferable in most instances to having to relink the database interface every time a program which uses it is linked.

```
2.3.3.2.1.2.2.3 Component Testing (*)
2.3.3.2.1.2.2.3.1 Coverage/Frequency Analysis
2.3.3.2.1.2.2.3.2 Mutant Analysis
2.3.3.2.1.2.2.3.3 Mutant Generation
```

Component Testing -- the third and final subphase of Component Engineering and the one that provides capabilities for assuring that a component works as intended.

Coverage/Frequency Analysis -- the capability to determine and assess measures associated with the invocation of program structural elements to determine the adequacy of a test run.

Mutant Analysis -- the capability to apply test data to a program and its mutants (i.e., programs that contain one or more likely errors) to determine test data adequacy.

Mutant Generation -- the capability to generate mutant programs by seeding a given program with common errors.

```
2.3.3.2.1.2.2.3.4 Target Support
2.3.3.2.1.2.2.3.4.1 Host-Target Upload-Download
2.3.3.2.1.2.2.3.4.1.1 Data
2.3.3.2.1.2.2.3.4.1.2 Executable Image
2.3.3.2.1.2.2.3.4.1.3 Object Code
2.3.3.2.1.2.2.3.4.1.4 Source Code
2.3.3.2.1.2.2.3.4.2 Target Code Generation
```

Target Support -- the set of capabilities necessary to allow code developed on one machine to be transferred to the machine where it will be executed, and the capabilities to retrieve test results and data from the target machine.

Host-Target Upload-Download -- the capability to transfer information between the development machine (i.e., the host) and the machine on which the system is to be executed (i.e., the target).

Data -- the capability to transfer data between the host and the target (e.g., test data and test results).

Executable Image -- the capability to transfer prelinked, executable programs to the target.

Object Code -- the capability to transfer machine level code to the target for linking and loading.

Source Code -- the capability to transfer source code to the target for compilation, linking, loading, execution.

Target Code Generation -- the capability to generate code on the host for execution on the target.

```
2.3.3.2.1.2.2.3.5 Test Case/Data Generation
2.3.3.2.1.2.2.3.5.1 Boundary Case Testing
2.3.3.2.1.2.2.3.5.2 Expected Output Generation
2.3.3.2.1.2.2.3.5.3 Functional Testing
2.3.3.2.1.2.2.3.5.4 Stress Testing
2.3.3.2.1.2.2.3.5.5 Structural Testing
2.3.3.2.1.2.2.3.5.6 Test Case Specification
```

Test Case/Data Generation -- the capability to automatically generate test cases and data for those test cases based upon the source code to be tested.

Boundary Case Testing -- the capability to generate test cases and data which will test the extremes (the boundaries) of a program.

Expected Output Generation -- the capability to generate the expected output from a testable program unit for later comparison with the actual output.

Functional Testing -- the capability to test normal control paths with normal, expected data.

Stress Testing -- the capability to test a program to see how it reacts to stress (e.g., multiple recursive invocations, devices off-line, degraded processor performance).

Structural Testing -- the capability to test a program's control structures, usually by executing all paths in the program or by selectively executing parts of every major path.

Test Case Specification -- the capability to generate test case specifications from testable units, including recording of test scripts, scenarios, expected output, etc.

```
2.3.3.2.1.2.2.3.6 Test Execution
2.3.3.2.1.2.2.3.6.1 Initial Testing
2.3.3.2.1.2.2.3.6.2 Regression Analysis
2.3.3.2.1.2.2.3.6.3 Regression Testing
2.3.3.2.1.2.2.3.7 Test Harness Generation
```

Test Execution -- the capability to manage and perform the execution of tests.

Initial Testing -- the capability to manage and control testing of units.

Regression Analysis -- the capability to indicate those units needing retesting by analysis of problem reports, change requests, test results, etc.

Regression Testing -- the capability to manage and control retesting of units based upon the results of the regression analysis.

Test Harness Generation -- the capability to produce a program that provides input to and encapsulates output from a testable program unit.

```
2.3.3.2.1.2.2.3.8 Test Result Management
2.3.3.2.1.2.2.3.8.1 Actual and Expected Data Comparison
2.3.3.2.1.2.2.3.8.2 Data Analysis
2.3.3.2.1.2.2.3.8.3 Data Reduction
2.3.3.2.1.2.2.3.8.4 Logging
2.3.3.2.1.2.2.3.8.4.1 Configuration
2.3.3.2.1.2.2.3.8.4.2 History
2.3.3.2.1.2.2.3.8.4.3 Result
2.3.3.2.1.2.2.3.8.5 Result Reporting
                        (see 2.3.3.2.4.1 Component Quality Evaluation)
```

Test Result Management -- the capability to record and manipulate test results.

Actual and Expected Data Comparison -- the capability to record actual test results against expected test results.

Data Analysis -- the capability to analyze data by statistical or other means, often implying the capability to "smooth" data.

Data Reduction -- the capability to perform data reduction (e.g., by statistical analysis) and data summarization.

Logging -- the capability to record test results and information about the system being tested.

Configuration -- the capability to record the software and hardware configurations in place when a given unit is tested.

History -- the capability to capture date and time stamps, which units were tested, which tests were executed, etc.

Result -- the capability to capture test output.

Result Reporting -- the capability to format test and collapsed data to produce test reports.

```
2.3.3.2.2 Prototype Construction (*)
2.3.3.2.2.1 Framework Specification (*)
              (see 2.3.3.2.1.2.2.1 Specification Development)
2.3.3.2.2.2 Framework Implementation and Component Hookup (*)
              (see 2.3.3.2.1.2.2.2 Implementation Development)
2.3.3.2.2.3 Prototype Exercising and Informal Testing (*)
              (see 2.3.3.2.1.2.2.2.3 Debugging)
              (see 2.3.3.2.1.2.2.3 Component Testing)
```

Prototype Construction -- the second of four sequential subphases of the Prototyping subphase, this subphase includes the capabilities for assembling a prototype from various components.

Framework Specification -- the first of three subphases of Prototype Construction. This subphase provides for creating (or instantiating) the external interface of the prototype.

Framework Implementation and Component Hookup -- the second of three subphases of Prototype Construction. This subphase provides for creating the implementation dependent portions of the prototype and hooking ("with"ing) the various components in.

Prototype Exercising and Informal Testing -- the third of three subphases of Prototype Construction. This subphase provides for the developers satisfying themselves that the prototype works as they had intended.

```
2.3.3.2.3 Prototype Evaluation (*)
2.3.3.2.3.1 User Interface Analysis
2.3.3.2.3.1.1 User Action Tracking
2.3.3.2.3.1.2 User Error Tracking
2.3.3.2.3.1.3 User Feedback Capture
2.3.3.2.3.1.4 Real-Time UI Modification and Resumption
```

Prototype Evaluation -- the third of four sequential subphases of the Prototyping subphase, this subphase includes the capabilities for assessing how well the prototype meets the developers' and the customer's needs.

User Interface Analysis -- the capability to collect and analyze data about a system's user interface(s).

User Action Tracking -- the capability to log each action a user takes for further analysis.

User Error Tracking -- the capability to log each error a user makes for further analysis.

User Feedback Captur^e^ -- the capability for the user to record notes, comments, and suggestions during a session on the -ototype system.

Real-Time UI Modification and Resumption -- the capability to modify the user interface at the customer's suggestion and immediately resume use of the modified interface.

```
2.3.3.2.4 Component Productization (*)
2.3.3.2.4.1 Component Quality Evaluation (*)
2.3.3.2.4.1.1 Static Analysis
2.3.3.2.4.1.1.1 Complexity Measurement
2.3.3.2.4.1.1.2 Completeness Checking
2.3.3.2.4.1.1.3 Consistency Checking
2.3.3.2.4.1.1.4 Cross Reference
2.3.3.2.4.1.1.4.1 Reference Only
2.3.3.2.4.1.1.4.2 Uninitialized Variable
2.3.3.2.4.1.1.4.3 Unused Variable
2.3.3.2.4.1.1.4.4 Value Change
```

Component Productization -- the final of four sequential subphases of the Prototyping subphase, this subphase includes the capabilities for making a component fit for insertion into the reuse repository.

Component Quality Evaluation -- the first subphase of Component Productization. This subphase provides capabilities for evaluating a component against prescribed quality standards.

Static Analysis -- the capability to provide operations against an object that analyze the object without regard to the dynamic nature of the object (e.g., whether or not it will execute).

Complexity Measurement -- the capability to derive an indication of how complex an object is.

Completeness Checking -- the capability to check an object against a set of completeness criteria.

Consistency Checking -- the capability to check an object against a set of consistency criteria.

Cross Reference -- the capability to determine where and how an object is defined and used.

Reference Only -- the capability to determine where an object is referenced.

Uninitialized Variable -- for those objects that are program variables, the capability to indicate which have never been initialized.

Unused Variable -- for those objects that are program variables, the capability to indicate which have been defined but never been used.

Value Change -- the capability to determine where an object's value has the potential to be changed.

```
2.3.3.2.4.1.1.5 Data Flow Analysis
2.3.3.2.4.1.1.6 Exception Analysis
2.3.3.2.4.1.1.7 Programming Style Analysis
2.3.3.2.4.1.1.8 Statement Profiling and Analysis
2.3.3.2.4.1.1.9 Structure Checking
```

Data Flow Analysis -- the capability to perform a (graphical) analysis of the sequential patterns of definitions and references of data.

Exception Analysis -- the capability to perform an analysis of the sequential patterns of exception definition and use. For example, this analysis could indicate where exceptions would potentially be raised, but not handled; exceptions are defined, but not used; exceptions are raised, but not handled locally, etc.

Programming Style Analysis -- the capability to analyze a program unit's conformance to a set of project specified style guidelines.

Statement Profiling and Analysis -- the capability to document the frequency and distribution of various classes of statements.

Structure Checking -- the capability to detect structural flaws in a program, for example, improper loop nesting, logical contradictions, unreferenced labels, unreachable statements, and statements without successors.

```
2.3.3.2.4.1.2 Dynamic/Performance Analysis
2.3.3.2.4.1.2.1 Resource Utilization
2.3.3.2.4.1.2.2 Timing
2.3.3.2.4.1.2.2.1 Subprogram
```

Dynamic/Performance Analysis -- the set of capabilities to document and analyze the run-time behavior of a system.

Resource Utilization -- the capability to record and analyze resource use (e.g., processor, number of tasks, I/O units) during execution of a program.

Timing -- the capability to record and analyze times associated with the execution of a program (e.g., CPU time, elapsed time).

Subprogram -- the capability to record and analyze times associated with the execution of individual subprograms.

```
2.3.3.2.4.1.2.3 Tracing
2.3.3.2.4.1.2.3.1 Data Flow Tracing
2.3.3.2.4.1.2.3.2 Path Flow Tracing
2.3.3.2.4.1.2.3.3 Tracepoint Activation/Deactivation
2.3.3.2.4.1.2.3.4 Tracepoint Display
```

Tracing -- the capability to capture and monitor the historical record of the execution of a program.

Data Flow Tracing -- the capability to capture and monitor changes to the state of variables during the execution of a program.

Path Flow Tracing -- the capability to capture and monitor the historical record of the statements executed during the execution of a program.

Tracepoint Activation/Deactivation -- the capability to selectively start and stop tracing specific portions of a program.

Tracepoint Display -- the capability to display all active and inactive tracepoints.

```
2.3.3.2.4.2 Formal Component Testing (*)
              (see 2.3.3.2.1.2.2.3 Component Testing)
2.3.3.2.4.3 Component Refinement (*)
2.3.3.2.4.3.1 Optimization
2.3.3.2.4.3.2 Preamble Generation
2.3.3.2.4.3.3 Tuning
```

Formal Component Testing -- the second subphase of Component Productization. This subphase provides capabilities for assuring that a component works as intended.

Component Refinement -- the third subphase of Component Productization. This subphase provides capabilities for remedying any deficiencies found during quality evaluation or testing.

Optimization -- the capability to reduce the resources consumed by a component.

Preamble Generation -- the capability to record, in a header belonging to a component, that information necessary for the understanding, use, and reuse of the component.

Tuning -- the capability to improve execution speed by modifying specific parts of programs known to cause execution inefficiencies.

```
2.3.3.2.4.4 Repository Insertion (*)

2.3.3.3 Prototype Selection (*)
```

Repository Insertion -- the fourth subphase of Component Productization. This subphase provides capabilities for inserting the fully refined, documented components into the reuse repository for use in future efforts.

Prototype Selection -- the third of three subphases of Prototype Competition, this subphase provides the capabilities to select one of the competing prototypes as the system model.

2.4 Productization (*)

Productization is the fourth phase of the SFLC. In this phase, the final system is refined from the results of the prototypes.

```
2.4.1 System Quality Evaluation (*)
        (see 2.3.3.2.4.1 Component Quality Evaluation)
        (see 2.5.4 Reliability Analysis)
2.4.2 System Testing (*)
        (see 2.3.3.2.1.2.2.3 Component Testing)
2.4.3 System Refinement (*)
2.4.3.1 Optimization
        (see 2.3.3.2.4.3.1 Optimization)
2.4.4 Final User and System Documentation Compilation (*)
```

System Quality Evaluation -- the first of four subphases of Productization, this subphase provides the capabilities to evaluate of the system against prescribed quality standards.

System Testing -- the second of four subphases of Productization, this subphase provides the capabilities to assure that the system performs as intended by the customer.

System Refinement -- the third of four subphases of Productization, this subphase provides the capabilities to make those changes identified in the System Quality Evaluation and System Testing subphases (e.g., performance enhancements, making all code meet certain standards).

Final User and System Documentation Compilation -- the fourth of four subphases of Productization, this subphase provides the capabilities to compile, in final form, all system and user documentation produced during development of the prototypes along with any new documentation.

2.5 Operational Maintenance (*)

Operational Maintenance is the fifth and final phase of the SFLC. The system delivered at the culmination of the Productization phase is maintained during this phase. This phase does not contains minimal development capabilities. Significant development activity would cause a loop back into the Development Engineering or Productization phase.

```
2.5.1 Problem Reporting
2.5.1.1 Problem Report Analysis
2.5.1.2 Impact Analysis
2.5.2 Change Requesting
2.5.2.1 Change Request Analysis
2.5.2.2 Impact Analysis
```

Problem Reporting -- the capability to record error information, error correction activities and status information of each error.

Problem Report Analysis -- the capability to analyze problem reports to determine the validity of the reported problem and a corrective action.

Impact Analysis -- the capability to analyze the effects of specific changes to an object on all other dependent objects. The analysis is useful in determining which alternative solution to choose.

Change Requesting -- the capability to record requests, rationale, and associated data for changes to a system.

Change Request Analysis -- the capability to analyze, and record the results of analysis of, change requests to determine the necessity of the change, technical/economical impacts, and approach to accomplishing the change.

Impact Analysis -- the capability to determine the impact of proposed changes on the system.

```
2.5.3 Performance Analysis
     (see 2.3.3.2.4.1.2 Dynamic/Performance Analysis)
2.5.4 Reliability Analysis
2.5.5 Software Updating
2.5.5.1 Patching Software
2.5.5.2 Replacing Software
```

Performance Analysis -- the capability to capture, analyze, and report on data concerning the performance of the operational system.

Reliability Analysis -- the capability to capture, analyze, and report on data concerning the reliability of the operational system.

Software Updating -- the capability to update part of the existing system without installing a new system.

Patching Software -- the capability to selectively modify parts of the existing system.

Replacing Software -- the capability to selectively replace parts of the existing system.

3 Communication Capabilities

This section of the taxonomy discusses those capabilities that are essential for project team members to communicate with the computer system and with each other.

3.1 Help Facility

The Help Facility provides the capabilities of prompter, tutor, and resident knowledge base about the tools and interfaces available. Usually it must cater to a large spectrum of user ability levels.

```
3.1.1 Context-Sensitive
3.1.2 Log
3.1.3 On-line Documentation
3.1.4 Tutorial/Training
3.1.4.1 Controlled Command Execution
3.1.4.2 Tuturial History Capture
3.1.4.3 User Learning Evaluation
```

Context-Sensitive -- the capability of the Help Facility to determine the context of the user and tailor the available help information to the specific problem.

Log -- the capability to record all help messages a user requests during a logon session.

On-line Documentation -- the capability to retain system documentation on-line for retrieval by the user.

Tutorial/Training -- the capability to guide environment users through a series of planned lessons aimed at teaching how to use the environment.

Controlled Command Execution -- the capability for users to "try out" commands (with guidance) from inside the tutorial capability.

Tuturial History Capture -- the capability for the environment to capture and retain usage information about its tutorials, especially for understanding for which subjects users feel the need for tutorials and for "remembering" how far a user has previously progressed through a tutorial.

User Learning Evaluation -- the capability to assess how well a user has learned material presented.

3.2 Project Communication

Project Communication is the collective set of capabilities that allow members of a project to correspond with each other.

```
3.2.1 Demonstration Chart Preparation
3.2.2 Electronic Conferencing
3.2.3 Electronic Forum
3.2.2 Electronic Mail
```

Demonstration Chart Preparation -- the capability to prepare demonstration charts (i.e., foils, storyboards) from objects contained in the project repository.

Electronic Conferencing -- the capability for two or more users to correspond concurrently on-line.

Electronic Forum -- the capability for several users to maintain ongoing discussion on specific topics.

Electronic Mail -- the capability to send and receive messages to and from other system users.

3.3 User Interface

The User Interface (UI) is the collective set of capabilities which allow users to communicate with the computer system via a terminal.

```
3.3.1 Command Support
3.3.1.1 Aliasing
3.3.1.2 Building
3.3.1.3 Completion
3.3.1.4 Confirmation of Destructive Action
3.3.1.5 Contexts
3.3.1.6 Error Recovery
3.3.1.7 Feedback
3.3.1.8 History Log
3.3.1.9 Procedures
3.3.1.10 Stacking and Retrieval
3.3.1.10 Undo
```

Command Support -- the set of capabilities that support the user in providing instructions to the software engineering environment.

Aliasing -- the capability of the user interface to allow the user to define and use synonyms for a command.

Building -- the capability of the user interface to guide the user through command completion. Guiding mechanisms could be promts, menus etc....

Completion -- the capability of the user interface to supply the remainder of a command based upon the user's partial input.

Confirmation of Destructive Action -- the capability that prompts the user to confirm a request to perform an action which would potentially be harmful or destructive (e.g., deletions, editing a high level unit).

Contexts -- the capability of the user interface to focus the user on a task by eliminating all nonessential information from view.

Error Recovery -- the capability of the user interface to guide the user through a series of corrective actions after an error has occurred.

Feedback -- the capability for a user interface to inform the user of the success or failure of every action.

History Log -- the capability to record all the commands a user invokes during a logon session.

Procedures -- the capability of the user interface to support writing and executing programs of commands (e.g., to alleviate repetition).

Stacking and Retrieval -- the capability by which the last 'n' number of commands are retained and can be retrieved by the user.

Undo -- the capability for a user interface action to be reversed or to make the state of the system such that that action had never occurred.

```
3.3.2 Interaction Modes
3.3.2.1 Command Line
3.3.2.2 Graphical
3.3.2.3 Menu
3.3.2.4 Object-Oriented
3.3.2.5 Pointing Device
3.3.2.6 Windowing
```

Interaction Modes -- the set of capabilities that provide for communication of user desires to the computer system.

Command Line -- the capability for the user to provide input in response to a command prompt.

Graphical -- the capability for the user interface to display system objects graphically and for the user to provide graphical input.

Menu -- the capability for the user to provide input by making a selection from a list.

Object-Oriented -- the capability for the user interface to display objects and to allow the user to select operations to act on those objects, as opposed to a user interface capability which supplies objects to operations.

Pointing Device Support -- the capability to support a pointing device such as a mouse, pen, touch sensitive screen.

Windowing -- the capability to support several simultaneous displays.

4 Global Object Manipulation Capabilities

This section of the taxonomy discusses those basic capabilities that are globally necessary in the viewing, modifying, and arranging of objects in a software engineering environment.

4.1 Browsing

Browsing is the capability to view an object without the authority to change it.

```
4.1.1 Structured Walkthrough
```

Structured Walkthrough -- the capability to view and traverse an object and related objects based upon it's structure (e.g., syntax, definition, relationships).

4.2 Comparison

Comparison is the capability to determine whether or not two objects are equivalent (and the capability to define equivalence).

4.3 Editing

Editing is the capability to selectively revise an object.

```
4.3.1 Graphics
4.3.2 Text
4.3.2.1 Screen
4.3.2.2 Line
4.3.2.3 Semantic Completion
4.3.2.4 Syntax Assistance
```

Graphics -- the capability to selectively revise graphical objects.

Text -- the capability to selectively revise textual objects.

Screen -- the capability to view and selectively revise textual objects a screen at a time.

Line -- the capability to view and selectively revise textual objects line by line.

Semantic Completion -- the capability of the editor assist entry of objects by providing input based on the user's partial input and on the semantics of the object being input.

Syntax Assistance -- the capability of the editor assist entry of objects by providing input based on the user's partial input and on the syntax of the object being input.

4.4 Formatting

Formatting is the arrangement of an object according to predefined and/or user defined conventions.

4.5 Output

Output is the capability to generate various forms of an object for use on various devices.

```
4.5.1 Graphics
4.5.1.1 File
4.5.1.2 Hardcopy
4.5.1.3 Screen
4.5.2 Graphics Interchange Format
4.5.3 Object Interchange Format
4.5.4 Text
        (see 4.5.1 Graphics)
```

Graphics -- the capability to generate graphical forms of an object.

File -- the capability to generate graphical forms of an object suitable for on-line storage and retrieval.

Hardcopy -- the capability to generate graphical forms of an object suitable for output on a hardcopy device (e.g., graphics printer. plotter).

Screen -- the capability to generate graphical forms of an object suitable for display on a terminal.

Graphics Interchange Format -- the capability to generate graphical objects in a format suitable for transfer to other software engineering environments.

Object Interchange Format -- the capability to generate non-graphical objects in a format suitable for transfer to other software engineering environments.

Text -- the capability to generate textual output suitable for on-line use, hardcopy, and terminal use.

4.6 Searching

Searching is the capability to identify (and retrieve) those objects from the software engineering environment which meet certain predefined or user-defined criteria and attributes.

4.7 Sort/Merge

Sorting is the capability to arrange objects in a specific order (e.g. alphabetical order). Merging is the sorting of two or more sets of objects into a single set of objects.